Professional Portfolio Application Proposal

Second Milestone - Software Design - INSO 4117

Prof. Marko Schütz-Schmuck

Table of Contents

1. Informative Part . . . 2
   1.1. Team . . . 2
   1.2. Current Situation, Needs, Ideas . . . 3
   1.3. Scope, Span, and Synopsis . . . 5
   1.4. Other Activities than Just Developing Source Code . . . 6
   1.5. Derived Goals . . . 7
2. Descriptive Part . . . 9
   2.1. Domain Description . . . 9
   2.2. Requirements . . . 35
   2.3. Implementation . . . 57
3. Analytic Part . . . 67
   3.1. Concept Analysis . . . 67
   3.2. Validation and Verification . . . 68
4. Specific Class Topics . . . 72
   4.1. Communication and the Use of Language . . . 72
   4.2. Isolating the Domain & Expressing Models in Software . . . 80
   4.3. Life Cycle of Domain Objects . . . 88
   4.4. Refactoring Towards Deeper Insight and Breakthroughs . . . 91
5. Log Book . . . 96

Notation for Changes

Symbol	Function	Example
Highlight	Added	This text is new.
Underline	Paraphrased	This text has been modified.
~~Strikethrough~~	Removed	~~This text is no longer relevant.~~

Chapter 1. Informative Part

1.1. Team

For Milestone 1, the team was organized into four functional groups. These are Research, Functionality, Design, and Documentation; plus three Managers. In this context, all team members are considered partners. No external partners have been identified for this milestone; if any arise, their roles and responsibilities will be documented. The lists are presented in alphabetical order and indicate each group's Team Leader. The Tasks column is reserved for recording completed issues.

Managers:

Student	GitHub username	Tasks
Kaysha L. Pagan López	@Kay9876	
Juan R. Rivera Rodríguez	@JRRR0912	
Julian A. Vivas Gandarillas	@julivivas	

Developers:

Student	GitHub username	Tasks
Luis J. Cruz Cruz *(Team Leader)*	@LuisJCruz	#4, #16, #45, #69, #76, #98
Yandre Cabán Torres	@YandreCaban	#16, #35, #69, #72
Carlos A. Ferrer Hayes	@CarlosxFerrer	#16, #34, #69, #75
Osvaldo F. Figueroa Canino	@Osvaldoo1414	#16, #36, #60, #62, #69, #71, #91
Ariana P. Rodríguez Méndez	@arianrodz21	#37, #69, #73

Student	GitHub username	Tasks
Jean P. I. Sánchez Félix *(Team Leader)*	@JeanSanchezFelix	#2, #15, #23, #24, #26, #38, #45, #59, #76, #92, #96, #98
Carlos E. Cabán González	@Carloscaban	#24, #32, #33
Jahdiel E. Montero Alicea	@JadStelar	#29, #30, #31, #67
Kiara N. Pérez Rivera	@kiarancole	#38, #39, #47, #48, #60, #61, #63, #99
Pedro A. Rodríguez Vélez	@PedroRodz	#13, #14, #24, #26, #28, #38, #39, #41, #49, #52, #53, #57, #82
Alexa M. Zaragoza Torres	@alexamzt	#24, #25, #26, #27, #38, #39, #50, #51, #53, #54, #55, #56, #101

Student	GitHub username	Tasks
Diego Pérez Ganradillas *(Team Leader)*	@diegoperez16	#17, #18, #21, #45, #65, #76, #78, #93, #94, #98, #103
Axel A. Orta Félix	@Axl47	#20
Carlos Pepin Delgado	@carlospepin23	#19, #21, #40, #42, #43, #44, #83, #84
Ryan R. Vélez Villarrubia	@RyanVelez101	#66, #95

Student	GitHub username	Tasks
Horeb Cotto Rosado *(Team Leader)*	@horebcotto21	#6, #12, #45, #68, #76, #98
Samarys Barreiro Meléndez	@SamarysB	#7, #77
Fernando Castro Cancel	@fernan-castro	#10, #74
Abdiel Cotto Claudio	@abdielcotto	#11, #70
Naedra J. Feliciano Agostini	@Naedra	#8, #79, #97
Heribiell A. Rodríguez Cruz	@LightPolution	#9, #38, #53, #58

1.2. Current Situation, Needs, Ideas

1.2.1. Current Situation

Moving from school to a job can be tough for grads and companies. Lots of students feel like job hunting is cold and doesn't work well. They send out tons of applications and don't hear back. Forbes Advisor says almost 60% of job seekers feel like companies just disappear on them, which is annoying and makes them worried about their careers. But companies have problems, too. They get flooded with applications for starter jobs, and a lot of people aren't a good match. It takes up a lot of time and money to find the right people. This whole thing is inefficient and doesn't really show when a candidate and a company might actually be a good fit for each other's goals. It can be a pretty bad experience for everyone.

1.2.2. Need

Structural friction in early-career recruiting (lack of visibility, opaque communication, and operational overload) reveals needs that belong to domain actors, not to a platform. The needs are outlined below.

Applicant / Recent Graduates

- Be discoverable for roles that truly match skills, interests, eligibility, availability, and location.
- Receive timely, transparent feedback about interest and status to reduce ghosting and uncertainty.
- Ensure fair access to opportunities and maintain control over privacy when sharing work and personal data.

- Obtain guidance to translate coursework, projects, and soft skills into recruiter-trusted signals.

Employers / Recruiters

- Efficient triage toward candidates who meet must-have criteria and show mutual interest.
- Rich, trustworthy evidence of capability and professional culture.
- Predictable, compliant communication and scheduling to minimize drop-off and miscommunication.

Cross-Cutting Needs

- Mutual-interest signaling before deep engagement.
- Early expectation alignment on role scope, compensation range, work modality, and timeline.
- Low-friction coordination for first conversations and follow-ups.
- Trust and safety: identity assurance, respectful conduct, and clear reporting channels.

Project-Internal Enabling Needs

- A shared domain description and a baseline set of requirements so the team understands needs independently of any system-to-be.
- A consistent, ubiquitous language across analysis, design, and code to prevent concept drift.
- Plans for requirements, architecture, component design, implementation, and testing to support whatever idea is chosen later.

1.2.3. Ideas

We propose a three-part design focused on a personalized, efficient, and high-quality user experience. The foundation of this approach is a onboarding and profile system. The system would create two fundamentally different experiences based on the user, whether they are a recruiter or a candidate. The system will request only the most relevant information for each persona, such as portfolios for students or verifying company details for recruiters. The system will have an interface that avoids clutter and ensures the platform feels built for each user from their very first interaction. Making it easier and more inclusive without replacing the current infrastructure.

Once users are onboarded, the swiping mechanism would enhance the core matching process by moving beyond a simple binary decision. This means creating carefully designed cards that act as a information display. The profiles can have an simple view and a more detailed view. The key to this design is a hierarchy that is informed by user research and which surfaces key decision making data relevant to the user directly in the swiping interface to maximize informed matches without causing overload.

Finally, to ensure connections are meaningful and productive, the mutual Match connection and messaging gateway would unlock only after both parties have shown interest. Afterwards, the system would immediately facilitate the first message and it could include some kind of icebreaker or customizable openers. Furthermore, a dedicated inbox to keep users organized, allow for easy profile review, and potentially integrate scheduling tools, transforming a simple Match into a genuine gateway to opportunity.

1.3. Scope, Span, and Synopsis

1.3.1. Scope and Span

Scope

The project's scope is to develop a mobile application aimed at improving the connection between students and recruiters. The app will address issues with traditional job search platforms and career fairs, which are often impersonal and inefficient, leading to a lack of engagement and missed opportunities. The project will encompass several key areas:

- Domain Engineering: Analyze the current landscape of student-recruiter engagement, identifying pain points in job fairs, static job boards, and passive search platforms. The goal is to create a faster, more efficient, and more engaging way for students and recruiters to connect.
- Requirements Engineering: Define system requirements to enable students to showcase their skills, qualifications, portfolios, and preferences dynamically. Recruiters will also be able to display what their company is offering and looking for. Requirements will focus on improving job placement rates, event attendance, and reducing the time spent in the recruitment process. These requirements will be refined continuously using direct feedback from both students and recruiters.
- Software Architecture: The architecture will feature a mobile front-end with a swipe-based matching system, real-time notifications, and event integration. The back-end will connect with job boards, applicant tracking systems, career services, and on-campus event data to strengthen student-recruiter engagement.
- Software Design Process: The project will follow an iterative design and development process, beginning with a pilot test to evaluate performance and identify areas of improvement. User feedback will drive optimization of the user interface, swiping experience, and the matching algorithm.

Span

The project's span is focused on creating a scalable and user-friendly solution that streamlines the student-recruiter connection process. The app is designed to support efficient matching, real-time communication, and event integration.

- Specifics of the System: Students can create detailed profiles including videos, portfolios, and soft skills. Recruiters will also create company profiles that highlight roles, culture, and expectations. When both parties swipe right, they are notified of a Match and can begin communicating via chat or set up interviews. The app will also notify students about campus events that involve companies they have swiped right on, even if a Match has not occurred.
- Target Audience and Expansion: The initial span of the project involves a pilot test with a defined user base of students and recruiters. Expansion will include partnerships with recruiters, direct marketing to universities, and support for on-campus career fairs. Over time, the platform will expand to larger student and recruiter networks beyond the initial pilot.
- Methodology and Maintenance: The project will adopt an iterative methodology with regular update cycles guided by new technology trends and continuous user feedback. Effectiveness

will be tracked through key metrics such as app usage frequency, Match success rate, recruiter follow-up rate, event attendance, and user satisfaction. The cycle of feedback, optimization and scaling will ensure the app remains relevant and impactful.

1.3.2. Synopsis

The project aims to develop a mobile application that revolutionizes the way students and recruiters connect by addressing the inefficiencies and impersonal nature of traditional job search methods. By leveraging a swiping feature, similar to popular dating apps, the platform will facilitate dynamic and real-time interactions between students showcasing their skills and recruiters highlighting their opportunities. The app will integrate with existing job boards and career services, providing a seamless experience that enhances visibility, engagement, and mutual interest. Through iterative design, user feedback, and continuous optimization, the project seeks to create a scalable solution that improves job placement rates and reduces recruitment time and effort.

The project aims to develop a mobile application that modernizes how students and recruiters connect by addressing the inefficiencies of traditional job search methods. It uses a swiping-based interface to enable dynamic and real time engagement between students presenting their qualifications and recruiters offering opportunities. Throughout its lifecycle, the project will progress through several key phases: requirements engineering to define user and system needs, software architecture and design to establish the platform's structure, implementation of both front-end and back-end components, and systematic testing and validation to ensure reliability and usability. An iterative approach will be adopted, allowing feedback to refine requirements and improve design over time. The final goal is to deliver a scalable and efficient application that enhances job placement success, fosters meaningful recruiter to student connections, and maintains user centered quality across all development stages.

1.4. Other Activities than Just Developing Source Code

Projects are successful when coding is supported by planning, communication, and documentation. These activities keep the team organized and ready for future development.

- Although source code development is a top priority on this project, its success depends on several activities that extend beyond programming. Documentation plays a crucial role in keeping the project aligned, covering goals, requirements, architectural decisions, and detailed contributions. Well-maintained records make it easier for members to integrate into new teams and ensure stakeholders remain informed about progress, scope, and direction throughout the project.
- While the application is not yet developed enough for full-scale testing, it is still necessary to plan for quality assurance. This involves deciding how unit testing, integration testing, and usability evaluation will eventually be carried out once prototypes and code are available. In parallel, version control practices such as branching strategies, pull requests, and code reviews can already be defined so the team is prepared to manage collaboration effectively when development begins. These preparatory activities set the standard for a structured and reliable workflow.
- Project management and communication establish the general structure that ties everything together during this milestone. Setting clear milestone goals, assigning responsibilities, and

documenting meeting outcomes help the team stay organized and avoid confusion. Regular communication ensures that issues are identified and addressed early, while planning for security, privacy, and future phases prepares the project for ongoing development. By combining documentation, early planning for testing and version control, and strong management practices with the coding that will follow, the team lays the foundation for a successful project.

1.5. Derived Goals

~~The project's goals go beyond building an app, focusing on connecting students and recruiters effectively while guiding team efforts and decisions.~~

~~* From the current situation, needs, and proposed scope, several goals appear that extend beyond simply building a mobile application. First, the project aims to create a system that improves the connection between students and recruiters by promoting mutual interest, transparent communication, and efficient matching. Students should be empowered to present their skills, interests, and career aspirations in a way that recruiters can trust and evaluate fairly, while recruiters should be able to quickly identify candidates that align with both role requirements and company culture.~~

~~* In addition to these user-focused objectives, the project seeks to ensure usability, maintainability, and adaptability. The system should be easy to use and flexible, so it can improve based on user feedback and grow from a small pilot group to a wider audience. Security and privacy are also central goals, ensuring identity assurance, safe communication, and control over personal and company data.~~

~~* Finally, internal team goals include ensuring collaboration, clear documentation, and structured processes for version control, testing, and requirements management. By clearly following these internal practices, the team can stay coordinated, reduce errors, and build a solid base for sustained success as development progresses, and the platform grows.~~

A key outcome of this project is the emergence of several derived goals that extend beyond the primary function of matching recruiters and job seekers. These goals leverage the platform's unique data and position to create secondary, significant value.

One major derived goal is the generation of actionable market intelligence. The matching process naturally produces a rich stream of data on skills, salaries, and hiring trends. By systematically learning from this data, the platform can provide a "Career GPS" for job seekers, offering personalized guidance on skill gaps, real-time salary benchmarking, and potential career trajectories. For recruiters, this intelligence enables strategic talent acquisition by identifying hidden talent pools, providing competitive benchmarking on metrics like time-to-hire, and predicting emerging skill demands. This transforms the platform from a simple transactional tool into an indispensable, intelligent partner for all users, with the data also serving as a critical feedback loop for the platform's own strategic development and product roadmap.

A second derived goal involves creating a dynamic bridge between education and industry. The data generated provides a real-time map of the skills employers need versus the skills candidates possess. This allows the platform to become a vital feedback mechanism for universities and bootcamps, helping them validate and modernize their curricula to close specific skills gaps, such

as a lack of training in high-demand tools like Docker. For students, it empowers informed educational investment by highlighting which courses and skills will most effectively increase their employability. On a broader scale, this data can fuel regional economic development by helping cities identify critical talent shortages and create targeted workforce programs, ultimately building a stronger, more aligned local economy.

Finally, a crucial derived goal is to transform the job search from a black box into a guided journey for the candidate. The platform can leverage its collective data to demystify the process, providing unprecedented transparency. This includes empowering job seekers with data-backed salary insights that illuminate how specific skills impact pay, enabling confident negotiation. It also involves providing a realistic "mirror" to the market, showing candidates exactly how their profile measures up and pinpointing specific areas for improvement. By offering a "pulse" on typical hiring timelines and illuminating common career progression paths, the platform reduces the anxiety and uncertainty of job hunting. This shifts its value proposition from merely finding a job to becoming a trusted advisor for managing an entire career, thereby building immense user loyalty and trust.

Chapter 2. Descriptive Part

2.1. Domain Description

2.1.1. Domain Rough Sketch

NOTE	This is an unprocessed collection of notes, quotes, and observations from the domain (student-recruiter interactions).

Student (Abdul, Mechanical Engineering senior): “At last year’s fall job fair, I stood in line almost 40 minutes just to hand my résumé to a recruiter from Pratt & Whitney. The line wrapped around the Student Center atrium — people were sweating, holding folders, and trying to look composed. When I finally got to the table, the recruiter just smiled, took my résumé, and said they’d be in touch. I wasn’t sure if she’d even remember me by the end of the day.” → **Observation:** Waiting fatigue, impersonal exchanges, uncertainty about follow-up.

Recruiter (Corey, Lilly Pharmaceuticals): “We meet hundreds of students in just a few hours. By the end of the day, names and faces blur together. I jot quick notes on sticky labels: ‘Python — confident’, ‘Good communicator’, or sometimes just a checkmark. Later, when reviewing, I can’t always recall which student said what.” → **Observation:** Recruiters rely on quick, shorthand impressions; limited retention of individual details.

Student (Maria, Computer Engineering junior): “This year I wore new shoes — bad idea. I had a class right before the fair and another right after, so I kept rushing back and forth between buildings. By the time I came back to the fair after my thermodynamics class, it was packed. I could barely hear anything; someone even stepped on my heel while I tried to talk to a recruiter from Google. It’s so loud you have to almost shout to be heard.” → **Observation:** Physical fatigue and environmental noise affect both comfort and communication quality.

Recruiter (Daniel, local tech startup): “By noon, my voice is gone. I try to smile and look engaged, but it’s hard to remember each student’s story. We keep a spreadsheet open on the booth laptop — name, major, key skills — just enough to remember who’s who. If a student brings something visual, like a project or a QR code to their portfolio, it really helps.” → **Observation:** Recruiters value portfolio evidence over résumé lines; digital aids improve recall.

Student (Lina, Electrical Engineering sophomore): “I applied on Handshake, then on LinkedIn, and again on the company website. I never knew if anyone actually saw it. After two months, no one replied, and I started to wonder if my applications were just getting lost somewhere.” → **Observation:** Lack of transparency in the post-fair pipeline; unclear feedback loop between applicant systems.

Recruiter (Rafael, mid-sized design firm): “We usually contact students weeks later, but by then many already accepted other offers or stopped checking their student email. If the first batch of candidates looks good, we stop looking. There’s just too little time.” → **Observation:** Recruiting process favors early applicants; short attention window post-fair.

Student (Jorge, freshman): “I was nervous to talk to recruiters because I didn’t even know what their companies did. Some just told me, ‘come back next year when you have more experience.’ I

mostly went this year just to practice — to get used to talking." → **Observation:** Early-year students attend fairs for exposure rather than placement; recruiters focus on upperclassmen.

Recruiter (Helen, aerospace company): "A lot of students come unprepared — they don't know which positions are open, or what our company even does. We'd love to pre-screen them, maybe through a short survey or a skill-based filter before the fair starts." → **Observation:** Recruiters express need for pre-fair filtering and digital preparation workflows.

Student (Emilio, Software Engineering senior): "The fair feels like speed dating — you only get two minutes to impress, then they move on. It's hard to show personality or teamwork skills when the line behind you is ten people long." → **Observation:** Limited interaction window leads to superficial evaluation; "soft skills" remain under-assessed.

Recruiter (Isabel, consulting firm): "We hand out branded tote bags and pens — it's funny, but sometimes that's what students remember us by. When I reach out later, they'll reply 'Oh, you were the ones with the blue water bottles!'" → **Observation:** Brand association through physical tokens enhances recall; small sensory cues matter.

Student (Arjun, Industrial Engineering senior): "I wish I knew right away if I had a chance instead of waiting months. After the fair, you check your email every day, but mostly nothing happens." → **Observation:** Desire for instant or early feedback mechanisms post-interaction.

Recruiter (Paula, tech recruiter): "We often rehire interns we already know. There's trust there — we've seen their work. That's why sometimes new applicants don't get the same attention." → **Observation:** Prior familiarity skews fairness; existing relationships heavily influence hiring behavior.

Student (Elena, Business Administration major): "The fairs are always packed — the noise, the pushing, it's chaotic. I tried to ask a recruiter a question, but they just handed me a flyer and moved on. Later, I realized that the quieter booths in the back actually had time to talk." → **Observation:** Spatial arrangement and booth placement affect engagement quality.

Career Advisor (Lucia, UPRM): "We encourage students to bring a 'target list' of companies, but many still wander without a plan. On the other side, recruiters tell us they want better visibility into who's coming — maybe an advance roster of students and their profiles." → **Observation:** Advisors recognize preparation asymmetry; opportunities for platform pre-matching.

2.1.2. Terminology

NOTE

Each term below is derived from raw observations in the Domain Rough Sketch (2.1.1) and refined through concept analysis. Terms are presented with their classification (entity, actor, event, etc.), domain scope, and traceability to source observations. The derivation of each term is explicitly shown through annotations linking back to specific observations, quotes, and patterns from the rough sketch.

This structured approach ensures transparency in how our domain vocabulary emerged from real-world observations rather than being imposed artificially. When new terms are needed, they should follow this same pattern of clear derivation from documented domain phenomena.

Applicant

(entity, domain) A person, usually a student or recent graduate, pursuing professional chances. Candidates strive to highlight their abilities and credentials via their portfolios. *Derived from: "Students often rely on school provided career services for resume templates"; "Student: 'I applied through Handshake, LinkedIn, and the company website'"*

Employer

(entity, domain) An organization that owns openings and ultimately employs candidates. Multiple recruiters may represent the same employer during sourcing and selection. *Derived from: "Recruiter: 'We usually contact students weeks later'"; multiple references to company and employer context*

Recruiter

(actor, domain) A hiring professional acting on behalf of an employer to discover, evaluate, and engage candidates. A candidate may interact with several recruiters for the same employer. *Derived from: "We meet hundreds of students in a single afternoon"; "Some recruiters only target juniors and seniors"; "Recruiters: 'We prefer quick ways to identify students with the right skills'"*

Portfolio

(entity, domain) A collection of an applicant's work, projects, and achievements. Portfolios provide recruiters with evidence of professional skills.

Skill

(entity, domain) A demonstrated ability, either technical or interpersonal, that contributes to an applicant's professional profile.

Qualification

(entity, domain) An educational or professional credential (e.g., degree, certification) that indicates formal preparation or eligibility.

Work Modality

(entity, domain) The way in which work is performed, such as on site, remote, or hybrid. *Derived from: Students value flexibility in work arrangements; need for early expectation alignment on work modality*

Compensation Range

(attribute, domain) The expected or offered salary **plus applicable benefits** (e.g., equity/options, health plan, stipends, shuttle/transport, on-site meals). Considered between applicant and employer. *Derived from: "Early expectation alignment on role scope, compensation range, work modality, and timeline"; need for transparency in total compensation*

Location

(attribute, domain) The geographic context for a role or event (e.g., city/region/country) or "remote-eligible," used for discovery and filtering.

Start Date

(attribute, domain) The intended employment start date or window associated with an opening or offer.

Mutual Interest Signaling

(event, domain) The occurrence of both applicant and employer expressing interest, creating the basis for a potential connection.

Connection

(entity, domain) The relationship established upon mutual interest. Day-to-day interaction typically occurs between the applicant and the **recruiter** representing the **employer**; any employment outcome is with the employer.

First Conversation

(event, domain) The initial professional interaction after a confirmed connection, typically between an applicant and a **recruiter**; it may still be considered as leading to a connection with the **employer**. *Derived from: "Student: 'I'm nervous approaching a recruiter if I don't already know about the company'"; importance of structured initial interactions*

Interview Modality

(taxonomy, domain) The manner in which an interview is conducted (e.g., in-person, virtual). Serves as the parent concept for specific interview types. *Derived from: "Job fairs are often loud, crowded, chaotic"; need for flexible interaction formats*

Validity Period

(attribute, domain) The time window during which an offer remains actionable before it expires.

Clarifications

(process, domain) Bidirectional questions and answers to resolve ambiguities (scope, duties, timeline) without changing negotiated terms.

Adjustments

(process, domain) Mutually agreed changes to offer terms (e.g., title, start date, compensation range) prior to acceptance.

Accept

(decision/event, domain) The applicant's affirmative decision to proceed under the current offer within its validity period.

Decline

(decision/event, domain) The applicant's explicit decision not to proceed under the current offer.

Ghosting

(behavior, domain) The act of ceasing communication without notice, leading to inefficiency in the recruitment process. *Derived from: "Student: 'I never know if recruiters actually looked at my resume or if it went into a pile'"; lack of feedback and communication*

Identity Assurance

(behavior, domain) The process of verifying that participants are authentic and represent legitimate individuals. *Derived from: Need for "trust factor" mentioned by recruiters; importance of verified connections*

Recruitment Event

(entity, domain) A scheduled occasion, such as a job fair or networking session, where applicants and employers directly engage. *Derived from: Multiple references to job fairs; "At the job fair, I stood in line 40 minutes just to hand over my resume"*

Expectation Alignment

(behavior, domain) The process of clarifying and agreeing on key role aspects, including scope, compensation, timeline, and modality. *Derived from: "Recruiters say a lot of students come unprepared, don't know what positions are open"; need for upfront clarity*

Trust and Safety

(behavior, domain) The assurance that professional interactions occur under respectful conduct, secure data handling, and clear reporting mechanisms. *Derived from: Recruiters mentioning "trust factor" with known candidates; need for safe, professional interactions*

Feedback

(event, domain) Information shared between employer and applicant regarding application status, interest, or evaluation, enabling transparency. *Derived from: "Student: 'I wish I knew immediately if I had a chance instead of waiting months'"; need for timely updates*

User

(technical/authentication, domain) An authenticated account in the system. Each User is typed as either Candidate (e.g., Student) or Recruiter; avoid using "User" to describe domain roles.

Student

(subset, domain) A Candidate currently enrolled at a university/college. Used when context involves campus events, coursework, or student services.

Profile

(entity, domain) The core representation of a User in the system (typed as StudentProfile or RecruiterProfile). Distinct from a Profile Card used for swiping.

StudentProfile

(typed entity, domain) A Candidate's profile containing résumé, skills, preferences, portfolio items, and visibility settings. Identified by an immutable UUID.

RecruiterProfile

(typed entity, domain) A Recruiter's profile including employer association, role/title, sectors, location, and verification status.

Profile Card

(ui artifact, domain) Condensed, swipeable representation of a Profile shown in the Discovery Feed. *Derived from: "Recruiters: 'We prefer quick ways to identify students with the right skills'"; need for efficient profile scanning*

Discovery Feed

(experience, domain) A personalized deck of Profile Cards presented for evaluation. *Derived from: "After a while, names and faces blur together"; need for structured, paced discovery*

Swipe

(action, domain) The primary gesture to evaluate a Profile Card. Right-swipe = Like; left-swipe = Pass. *Derived from: "Students compare the process to 'speed dating'"; need for quick, clear interest signals*

Like

(action, domain) An expression of interest on a Profile Card (right-swipe). Stored by the system for Match evaluation.

Pass

(action, domain) A dismissal on a Profile Card (left-swipe). Removes the card from the current session.

Match

(event, domain) Created only when both sides have explicitly liked each other's Profile Cards (mutual interest signaling).

Message

(entity/action, domain) A communication exchanged only when a valid Connection exists (or explicit permission). *Derived from: "Informal hallway conversations sometimes lead to opportunities"; need for structured yet natural communication*

Opening (Job Opening)

(entity, domain) A role posted by an Employer with explicit Requirements, Location, Work Modality, Compensation Range, and Start Date. *Derived from: "Recruiters say a lot of students come unprepared, don't know what positions are open"; need for clear role definition*

Requirements

(structure, domain) Must-have and nice-to-have criteria for an Opening (e.g., skills, eligibility, language, authorization). Used to assess Eligibility. *Derived from: Recruiter notes like "Has Python", "Strong communication", "Not ready"; need for structured evaluation criteria*

Eligibility

(assessment, domain) Whether a Candidate meets the Requirements of an Opening (meets / partially meets / does not meet).

Shortlist

(collection, domain) A curated set of Candidates selected by a Recruiter for next steps (review, outreach, interview).

Interview

(event, domain) A scheduled conversation following a Connection/Shortlist; must respect non-overlapping time blocks and uses an Interview Modality.

RSVP

(action/state, domain) An explicit intent to attend an Event; updates capacity and powers reminders.

Offer

(entity, domain) A proposal from an Employer to a Candidate with explicit terms (role, Compensation Range, Location/Work Modality, Start Date) and a Validity Period.

Notification

(system event, domain) An in-app alert for key events (e.g., Match created, unread Message, Event reminder, Offer updates).

Visibility

(setting, domain) Exposure level of a StudentProfile: Public (searchable), By Match (visible only to the matched party), or Private (not discoverable; shared explicitly).

Session

(technical, domain) The authenticated runtime context for a User. Authorizes actions (swipes, messages, RSVPs).

2.1.3. Domain Terminology in Relation to Domain Rough Sketch

- Recruiter: (Actor) A user who represents a company or organization and uses the platform to discover and evaluate potential candidates. This term was refined from the rough concept of employer to specify the human actor, distinct from the company entity itself.
- Candidate: (Actor) A user who is seeking professional opportunities and uses the platform to discover companies and recruiters by reviewing recruiter profiles. This term should be revised as a candidate alludes to the user's primary role once a connection is made.
- Profile: The core digital representation of a user within the system. It is categorized as either a candidate profile or a recruiter profile containing corresponding relevant information.
- Swipe: The primary gesture of evaluation. A right swipe indicates a Like or expression of interest, and a left swipe indicates a Pass, which indicates a dismissal. This defines the core action of the system, removing ambiguity from the informal "Tinder-like" description.
- Profile Card: The user interface component that presents a condensed view of a Profile within the feed for the purpose of being swiped on. This term distinguishes the interactive element from the full Profile data structure.
- Match: A domain event that occurs only when two users have mutually Liked each other's profile cards.
- Connection: The persistent relationship state between two users that is established upon a Match. This term defines the context in which messaging and further interaction can occur. Moving beyond the transient event of the Match itself.
- Discovery Feed: The main application view where a user is presented with a "deck" of profile cards to evaluate. This term provides a specific name for the core screen, derived from its purpose.
- Student: A person currently enrolled at a university/college. In our domain, every Student is a Candidate, but not every Candidate is a Student. We use **Student** when context involves campus events, student portfolios, or coursework.
- User: A technical/authentication concept. A logged-in account that is either a **Student** or a

Recruiter. We avoid using “User” to describe domain roles.

- Employer: The organization that a Recruiter represents. Owns job **Openings** and brand presence. Distinct from the person acting (Recruiter).
- Company (alias of Employer): The institutional profile representing the Employer in the system (logo, description, sectors, location).
- StudentProfile: Typed Profile for a Student/Candidate. Contains resume, skills, preferences, visibility settings, and **Portfolio** items.
- RecruiterProfile: Typed Profile for a Recruiter. Contains company association, role/title, sectors, location, and verification status.
- Portfolio: Evidence of work attached to a StudentProfile (projects, links, PDFs, media). Supports recruiter evaluation.
- Event: A scheduled activity relevant to recruiting (career fair, info session, meetup). Used for discovery, RSVP, and attendance tracking.
- RSVP: An explicit intent to attend an Event. Updates capacity and powers reminders.
- Notification: A system alert delivered to an account (e.g., Match created, unread Chat, Event reminder).
- Queue:An ordered waiting line (physical or digital) used to preserve turn order (e.g., at a booth or for processing requests).
- Opening (Job Opening): A role published by an Employer with defined **Requirements** (must-haves, nice-to-haves), location, modality, and timeline.
- Requirements: Structured criteria for an Opening (skills, eligibility, language, authorization). Used to assess **Eligibility**.
- Eligibility: Whether a Student/Candidate meets the defined Requirements of an Opening (meets / partially meets / does not meet).
- Shortlist: A curated set of Candidates selected by a Recruiter for next steps (review, outreach, interview).
- Interview: A scheduled conversation between a Recruiter and a Student following a Connection/Shortlist. Must avoid overlapping time blocks.
- Visibility Profile exposure setting for a StudentProfile:
  - **Public** — appears in Recruiter search.
  - **By Match** — visible only to the matched party.
  - **Private** — not discoverable; shared only by explicit action.
- Identity (UUID): An immutable unique identifier assigned to core entities (Profiles, Matches, Events). Ensures stability across updates and systems.
- Session: The authenticated runtime context for an account. Authorizes actions (swipes, messages, RSVPs).
- VerificationPolicy A rule requiring a Recruiter/Employer to satisfy verification checks before certain actions (e.g., messaging, event hosting).
- Invariant: A rule that must always hold at the model boundary (e.g., no duplicate

Recruiter–Student Match pairs).

- Factory: A creation mechanism that enforces Invariants when instantiating entities (e.g., MatchFactory ensures valid parties and uniqueness).

2.1.4. Narrative

The domain concerns how employers identify, evaluate, and hire talent, and how job seekers explore and compare opportunities. In this context, the term “candidate” is used broadly to mean any person seeking a job, regardless of prior experience or stage of career. The observable world of the domain includes candidates with educational and employment histories, competencies, and preferences; employers with staffing needs and hiring calendars; recruiters who operate search and prioritization criteria; openings defined by requirements, number of positions, and relevant dates; application documents such as résumés, certifications, portfolios, and references; and recruitment events with agendas and limited capacity. Information circulates through common channels such as postings, professional networks, referrals, unions, and agencies, under rules of eligibility and time availability.

A typical cycle begins when an employer defines and disseminates a vacancy with explicit requirements, for example minimum experience, qualifications, licenses or certifications, languages, and legal authorizations, together with job conditions such as work modality, location, reference compensation, and start date. Candidates discover these opportunities, compare requirements to their profile, and express interest. Reciprocally, recruiters may pursue profiles based on observable signals such as track record, achievements, work samples, or references. Eligibility verification precedes any advance. Based on that verification and perceived fit, shortlists are formed and, when appropriate, additional screenings are conducted, including technical or psychometric tests. Interviews are scheduled in non-overlapping time slots, held in person or virtually, and conclude with observations that inform later decisions.

When there is sufficient correspondence between needs and demonstrated capabilities, the employer issues an offer with explicit terms and a validity period. Clarifications or adjustments may follow, after which the candidate accepts or declines within the stated deadline. In parallel, recruitment events concentrate interactions: an audience is convened, capacity is managed, attendance is confirmed, and participation is recorded. From these contacts, new expressions of interest and applications can emerge without a prior formal submission. Relevant information, including requirements, application states, evaluations, schedules, and event capacity, shows varying degrees of structure, which explains asymmetries and information overload on both sides.

The domain is governed by stable rules: each search process ties recruiters to a specific employer; candidates and openings relate in a many-to-many manner through applications; applications do not advance without meeting minimum requirements; no person is assigned to overlapping interviews; attendance at an event consumes available capacity; and offers expire if no response is received within their validity period. Cyclical patterns also exist, with posting and closing windows that shape supply, participation, and acceptance decisions. This description reflects how the labor market operates, independent of any system to be built.

Scope and actors

The domain covers how recruiters discover candidates, evaluate evidence of fit, and make time-

bound decisions. It also covers how students prepare and publish profiles and artifacts, apply to openings, communicate with recruiters, and respond to decisions. Primary actors are students, recruiters, and organizations. Secondary actors are career offices and third-party services that send notifications or store artifacts.

Core flow of a hiring cycle

1. A recruiter defines an opening with role, eligibility, skills, seniority, location rules, and a clear decision calendar.
2. Students prepare a profile and publish artifacts such as resume, projects, certifications, and availability.
3. Students submit an application to an opening. The application freezes the versions of the artifacts used for that opening.
4. Recruiters triage the queue of applications using quick signals such as eligibility, program, graduation term, skills match, portfolio completeness, and recent activity.
5. Selected students move to screening and interviews. Interview outcomes and notes accumulate as evidence tied to the same application.
6. Recruiters decide. Outcomes can be rejection, waitlist, or offer. An offer specifies deadline, compensation ranges or bands, start date window, and required actions.
7. Students accept, decline, or ask for more time. The system records a final state for the application and closes the loop with both sides.

Key entities and relationships

Module Organization (Lecture Topic Task: Modules for Profile and Matching)

To improve maintainability, domain clarity, as well as alignment with the principles of Domain Driven Design, the system has been organized into two primary modules: **Profile** and **Matching**. These modules group related classes and behaviors together, each with a clearly defined responsibility within the application domain.

Profile Module

This module encapsulates all functionality related to user identity and profile management. It defines how both students and recruiters represent themselves within the system and provides mechanisms for storing, displaying, and updating profile data.

Included Classes: `StudentProfile` `RecruiterProfile` `Resume` `MediaUpload` `ProfileController`

Core Responsibilities:

- Handle the creation, modification, and persistence of profile data for all users.
- Manage resume and media uploads associated with each profile.
- Ensure that user information is stored and retrieved consistently across the application.

- Provide limited, read only access to profile data through a defined interface, `IProfileReader`, for external modules such as Matching.

The Profile module acts as the source of truth for user related data and serves as an independent subsystem that can evolve without affecting other parts of the domain.

Matching Module

The Matching module defines the behavior that connects students and recruiters through an intelligent matching process. It consumes data from the Profile module to perform its operations but maintains its own internal logic and data flow.

Included Classes: `Match` `MatchService` `MatchingPolicy` `RecommendationEngine`

Core Responsibilities:

- Execute the matching algorithm that pairs students with recruiters based on skill, preference, and availability criteria.
- Manage match creation, updates, and persistence.
- Notify users when a match occurs and handle match related interactions.
- Implement and refine matching policies that determine how compatibility is calculated.

The Matching module does not modify or persist profile data directly. Instead, it retrieves read only information from the Profile module through the `IProfileReader` interface to maintain a clean separation of concerns.

Module Boundaries and Dependency Flow

To maintain loose coupling and modularity, the dependency flow between modules follows a one directional structure: Profile → exposes → IProfileReader, while Matching → uses → IProfileReader.

This ensures that the Matching module depends only on the data abstractions provided by Profile, while Profile remains independent of Matching. As a result, changes in the matching logic do not affect the internal behavior of the Profile module.

Benefits of Modular Design

This modular separation strengthens system scalability and future adaptability. For example:

- The Profile module can evolve to include new profile types or attributes without affecting matching logic.
- The Matching module can introduce new algorithms or policies without impacting profile management.
- Code review and testing processes become simpler due to clearer ownership boundaries.

Through this structure, the system adheres to the principles of high cohesion and low coupling, making sure that each part of the domain remains understandable, testable, and independently maintainable.

Relationship	Multiplicity	Notes
Student to Application	one to many	A student may submit many applications. An application belongs to one student.
Opening to Application	one to many	An opening receives many applications. An application targets one opening.
Application to Interview	one to many	Each interview is tied to one application and records stage, outcome, and notes.
Application to Offer	zero or one	At most one active offer per application. Historical offers remain as records.
Offer to Acceptance	zero or one	One acceptance closes the offer. Decline also closes the offer.
Student to Artifact	one to many	Artifacts are versioned. An application references the versions used at submit time.
Recruiter to Opening	one to many	A recruiter may own several openings across teams or time.
Notification to Event	many to one	Multiple notifications can be sent for a single domain event with different channels.

Invariants that guide design

- An application always links to exactly one student and one opening.
- An offer cannot exist without an application in a decision-eligible state.
- Once an offer is accepted, the application moves to hired and no other offers can be issued for that application.
- Deadlines are stored with time zone and source. Any change to a deadline keeps a trace of who changed it, when, and why.
- Interview outcomes and notes are immutable records once submitted. Corrections are stored as new records that supersede older ones.
- Notifications are reproducible. Given an event and a preference set, the system can explain which messages went out, to whom, and when.

Concrete examples from raw observations

- First triage by recruiters often takes less than one minute and checks basic eligibility and red flags such as missing graduation date or visa requirement.

- Students reuse the same resume across many openings. The application must keep the exact file seen during triage even if the profile later changes.
- Interviewers rely on a daily view named Interview Today with candidate, role, time window, location or link, and a quick link to notes.
- Offers require reminders at common timing windows such as T-24 hours and T-2 hours before the deadline.
- Career offices request an audit record of all messages sent to a student, including channel and delivery status.

Edge cases and ambiguity resolution

- A student accepts after the deadline because a recruiter granted an extension by email. The system records an extension event with the new limit and the actor who granted it.
- A recruiter publishes an offer with a deadline that is too early. The correction updates the active deadline and preserves the original as an error record. All related reminders are recalculated.
- A student submits two applications to the same opening through different channels. Duplicate detection flags the situation and asks the recruiter to merge or keep separate with a reason.
- An opening is withdrawn after interviews due to budget freeze. All active applications move to closed by employer with a reason code and a message to candidates.
- A student withdraws an application after receiving an external offer. The application state becomes withdrawn by candidate and future notifications stop.
- A student updates a resume after applying. The application still shows the submitted version and also displays that a newer profile exists for transparency.
- Recruiter reassignment happens mid-process. Ownership moves to a new recruiter while preserving the decision trail and permissions on notes.

Language and abstractions used consistently

- Ubiquitous terms include Student, Opening, Application, Interview, Offer, Acceptance, Reminder, and Notification.
- Application is the aggregate root for interviews, offer, acceptance, and decision notes. All changes that affect the decision state go through the application.
- Offer Deadline is a value that carries time zone and precision to minutes.
- Artifact Version captures the exact resume or portfolio snapshot attached to an application.
- Triage View and Interview Today are application services that orchestrate domain data into the screens recruiters use.

Why this structure matters?

This narrative ties the abstract model to observable work. The multiplicities clarify what can exist at the same time. The invariants prevent silent corruption such as orphaned offers or moving deadlines without trace. The edge cases show where business rules bend and how the system

should keep truth and history. The language aligns with the rough sketch and the terminology so that design, code, and tests refer to the same concepts.

2.1.5. Events, Actions, and Behaviors

Events

Event	Who/What triggers it	Immediate responses	Expected outcome/postcondition	Anchor to Sketch
Recruiting event announced or openings published	Recruiter, University, or Organizer	Recruiter defines roles and requirements; the organizer or university disseminates the information	Candidates learn about opportunities and plan attendance or apply online	Job fairs; desire to pre-screen before the fair
Arrival and check-in at the event such as a campus fair, industry fair, or meetup	Candidate or Student	Candidate registers and receives the map and agenda	Candidate is able to approach booths and tables	Fairs are loud, crowded, and chaotic
Queue formed in front of a booth or table	Candidates	Candidate waits; staff organizes the line	Long wait times and very short turns	Reports of waiting up to forty minutes
Brief booth interaction / quick pitch	Candidate and Recruiter	Candidate delivers a 30 to 60 second elevator pitch; recruiter asks quick questions	Initial and superficial evaluation	Interactions resemble speed dating and create stress on both sides
Resume or portfolio handoff, physical or digital	Candidate	Candidate hands over the résumé or portfolio; recruiter receives and sorts it	The document enters a pile or list and may lose visibility	Concern that résumés are not actually reviewed
Recruiter note taking	Recruiter	Recruiter records short notes and tags on paper or in a spreadsheet, for example: has Python; not ready	Notes are linked to the contact for later review	Many recruiters rely on spreadsheets for quick notes
Informal hallway or meetup conversation	Candidate and Recruiter	They meet away from the booth and talk informally	May create a lead or opportunity	Informal conversations sometimes lead to opportunities

Event	Who/What triggers it	Immediate responses	Expected outcome/postcondition	Anchor to Sketch
Application submitted through multiple portals or via referral	Candidate	Candidate applies through Handshake, LinkedIn, the company website, or by referral	Parallel entries for the same profile	Uncertainty about whether multichannel applications are reviewed
Pre-screen or quick sift	Recruiter or System	Recruiter uses heuristics such as year, visible skills, and seniority; the system applies rules	Candidate is marked preliminarily eligible or ruled out by a documented rule	Preference for quick identification of fit and targeting juniors or seniors
Delayed invitation or follow up	Recruiter	Recruiter contacts candidates weeks later	Some candidates are no longer available or interested	Late outreach leads to loss of interest
Decision to stop sourcing	Recruiter	Recruiter stops the search when the first batch looks good	The window for new applicants narrows	Search ends once early candidates meet expectations
Rehire of known talent such as former interns	Recruiter	Recruiter prioritizes known profiles	Faster hiring due to a trust factor	Rehiring former interns is common
Swag or promotional material handed out	Recruiter	Recruiter distributes swag	Improves employer brand recall	Candidates often remember the company by the swag

Actions

Candidate. Before showing up, the candidate tunes the résumé and, if applicable, the professional portfolio to the role and researches the company. #When it is their turn, they deliver a brief elevator pitch, hand over a résumé, and ask specific questions. In parallel, they submit applications through one or more platforms and keep a courteous follow up thread by email or LinkedIn. If responses stall or plans change, they may withdraw or pause the application.

Behaviors

In person recruiting flow.

The process moves from announcement to attendance, then queuing, a brief conversation at the booth, and follow up. Noise, crowding, and time pressure push interactions toward quick

impressions and coarse screening. Soft skills are hard to judge in under a minute.

Candidate engagement and application behavior

This behavior encompasses the end-to-end process through which a candidate interacts with the recruiting process. It begins when the recruiting event is announced and the candidate adjusts their resume and portfolio to match desired roles (action). Upon arrival, the candidate checks in (event) and approaches booths to deliver a brief introduction (action). They hand over a resume (event) and often follow up by submitting applications through multiple portals (action). After the event, they may send follow-up messages or withdraw applications (action). These linked steps form a behavioral arc that involves preparation, event participation, and post-event actions. * Reference to Sketch:* Students prepare portfolios, deliver pitches, hand over resumes, and apply across multiple systems.

Fast screening and prioritization.

To manage volume, recruiters rely on simple heuristics such as year, visible skills, and seniority, as well as rule based filters. Throughput improves, but the risk of overlooking strong profiles increases.

Recruiter sourcing and screening behavior

This behavior represents how recruiters identify, evaluate, and narrow candidates. It starts when a recruiter defines a role and publishes an opening (action), triggering the recruiting event (event). Recruiters then attend, solicit introductions (action), and record notes (action) immediately after brief exchanges (event). They apply prescreen heuristics to sort through candidates (action) and may eventually decide to stop sourcing (action/event). * Reference to Sketch:* Recruiters rely on heuristics, jot notes on spreadsheets, and often end sourcing once a candidate seems sufficient.

Multichannel applications and unclear status. Candidates often apply through several portals and by referral. Without a single source of truth, duplicate records and uncertain statuses appear, which confuses both sides and slows review.

Multichannel application handling and consolidation

This behavior involves the candidate's repeated submissions across different systems and the institution's attempts to merge those inputs. A candidate submits multiple applications through various portals (event/action). The system removes the duplicates of these entries (action) and links records for traceability (action), often applying prescreening automatically (event). * Reference to Sketch:* Candidates apply via Handshake, LinkedIn, and company sites, causing redundant records and recruiter uncertainty.

Stopping after the first promising group. When the initial cohort seems sufficient, active sourcing pauses. The window for new applicants narrows and the range of options can shrink.

Communication delay and attrition. If outreach arrives weeks after the first contact, interest declines and other processes advance. Early signals and timely touchpoints reduce drop off.

Uneven candidate preparation. Some arrive with generic résumés or limited knowledge of the company, while others attend mainly to practice and build confidence. The level of preparation

shapes the quality of the pitch and the impression left.

Informal networking that converts well. Unplanned conversations away from the booth allow calmer and more genuine exchanges that sometimes outperform the formal interaction.

Informal networking conversion behavior This behavior occurs when informal, unplanned exchanges result in hiring outcomes. An informal hallway conversation happens (event), followed by a recruiter or candidate initiating a focused follow-up (action) and logging or scheduling an interview (action). * Reference to Sketch:* Informal conversations often produce stronger opportunities than planned booth exchanges.

Notes as the memory of the process.

With many brief encounters, concise notes and tags become essential to remember people and make decisions. Consistent record keeping improves later review and shortlist.

Rehiring based on trust. People already known to the team, such as former interns, are often prioritized because their performance is validated. Time to hire shortens and uncertainty decreases compared with external candidates.

Brand recall from presence and giveaways.

A well run booth and thoughtful materials strengthen memory of the employer and help reengage candidates after the event.

Recruiting event announced (event)

- Type: event
- Label: recruiting event announced
- Triggered by: Recruiter, University, or Organizer.
- Immediate response: Recruiter defines roles and requirements; organizer disseminates event details.
- Expected outcome/postcondition: Candidates learn about openings and plan attendance or apply online.
- Reference to sketch: Job fairs; desire to pre-screen before the fair.

Candidate checked in (event)

- Type: event
- Label: candidate just arrived and checked in
- Triggered by: Candidate or Student.
- Immediate response: Candidate registers and receives a map and agenda.
- Expected outcome/postcondition: Candidate can approach booths and recruiters.
- Reference to sketch: Loud, crowded, and chaotic fair environment.

Queue formed in front of booth (event)

- Type: event
- Label: queue just formed in front of a booth
- Triggered by: Candidate arrivals at a specific booth.
- Immediate response: Candidates wait; staff organizes the line.
- Expected outcome/postcondition: Long wait times and brief interactions.
- Reference to sketch: Reports of waiting up to forty minutes.

Recruiter solicited introduction (event)

- Type: event
- Label: recruiter just solicited introduction
- Triggered by: Recruiter action to request a pitch.
- Immediate response: Candidate delivers a 30–60 second elevator pitch; recruiter asks quick questions.
- Expected outcome/postcondition: Superficial first impression; rushed evaluation.
- Reference to sketch: “Speed dating” analogy; recruiter and candidate stress.

Resume/portfolio handed over (event / entity)

- Type: hybrid — event (handed over) and entity (resume/portfolio)
- Label: resume or portfolio handed over
- Triggered by: Candidate.
- Immediate response: Recruiter receives and sorts document (physical or digital).
- Expected outcome/postcondition: Document enters a review pile/list and may lose visibility.
- Reference to sketch: Concern that resumes are not reviewed.

Recruiter recording notes (operation / event)

- Type: operation (note-taking) / event (note recorded)
- Label: recruiter just recorded notes
- Triggered by: Recruiter following a candidate interaction.
- Immediate response: Recruiter writes short tags such as "has Python" or "not ready".
- Expected outcome/postcondition: Notes linked to contact for later review.
- Reference to sketch: Spreadsheets used for quick note taking.

Informal meetup occurred (event)

- Type: event
- Label: informal hallway or meetup conversation occurred

- Triggered by: Candidate and Recruiter (unplanned proximity).
- Immediate response: Unplanned exchange outside the booth.
- Expected outcome/postcondition: New lead or opportunity may emerge.
- Reference to sketch: Informal interactions often valued more than booth chats.

Multichannel application submitted (event / entity)

- Type: event (submission) / entity (application record)
- Label: application submitted through multiple portals
- Triggered by: Candidate.
- Immediate response: Candidate uploads resume to Handshake, LinkedIn, company website, or via referral.
- Expected outcome/postcondition: Duplicate entries for same profile; confusion about visibility.
- Reference to sketch: "Did they even see my application?"

Pre-screen completed (event)

- Type: event
- Label: pre-screen just completed
- Triggered by: Recruiter or screening system.
- Immediate response: Recruiter/system applies heuristics (year, skills, seniority) or filters.
- Expected outcome/postcondition: Candidate marked eligible or ruled out based on preset rules.
- Reference to sketch: Desire for quick identification of fit candidates.

Invitation/follow-up delayed (event)

- Type: event (delay manifested)
- Label: invitation or follow-up has been delayed
- Triggered by: Recruiter or resourcing backlog.
- Immediate response: Recruiter contacts candidate weeks later.
- Expected outcome/postcondition: Candidate may have lost interest or accepted another offer.
- Reference to sketch: Late outreach leads to attrition.

Decision to stop sourcing (event / operation)

- Type: event (decision made) / operation (halt sourcing)
- Label: decision to stop sourcing was just made
- Triggered by: Recruiter evaluation of candidate pool.
- Immediate response: Recruiter halts search once initial candidates seem sufficient.
- Expected outcome/postcondition: Window for new applicants narrows; reduced diversity of options.

- Reference to sketch: “If the first batch looks good, we stop looking.”

Known talent rehired (operation / behavior)

- Type: operation (re-hire) and recurring pattern (behavior)
- Label: known talent has been rehired
- Triggered by: Recruiter preference for prior interns/employees.
- Immediate response: Prior candidates prioritized; faster hiring due to trust.
- Expected outcome/postcondition: Reduced onboarding uncertainty; shorter time-to-hire.
- Reference to sketch: "Trust factor" in re-hiring interns.

Swag distributed (event / entity)

- Type: event (distribution) / entity (swag)
- Label: swag or promotional material was handed out
- Triggered by: Recruiter or marketing team.
- Immediate response: Recruiter distributes branded items.
- Expected outcome/postcondition: Improved employer brand recall among candidates.
- Reference to sketch: Students remember companies by their swag.

Recruiting system / institution (system / entity)

- Type: system / entity
- Label: system/institution (registration, deduplication, filters)
- Capabilities: announce events, register check-ins, collect/store records, deduplicate multichannel entries, apply automated prescreening, maintain traceability/versioning.
- Reference to sketch: Institutional compliance and traceability.

Reference providers (third-party entity)

- Type: entity
- Label: third parties (referees, background-check providers)
- Role: submit references, return verifications, confirm candidate details.

Resume repository / application repository (entity / operation)

- Type: entity (repository) / operation (persistence)
- Label: transfer of application records into repository
- Role: store application instances, link duplicates, provide a source-of-truth for recruiter review.

Note-taking repository (entity)

- Type: entity

- Label: recruiter notes repository (spreadsheets/ATS notes)
- Role: persistent place for quick tags and recall.

Prescreen operation (operation)

- Type: operation
- Label: prescreen operation
- Role: apply heuristics and filters to candidate records (automated or manual).

Transfer logs (entity / artifact)

- Type: entity (log)
- Label: transfer logs (audit trail for operations)
- Role: record events triggered by operations (e.g., pre-screen applied, invitation sent).

Transfer repository (entity)

- Type: entity
- Label: transfer repository (persistent transfer entities)
- Role: if transfer is modeled as entity, make it persistent for querying and linking.

Candidate engagement (behavior)

- Type: behavior — sequence of actions and events
- Definition / justification: This is a multi-step behavioral arc composed of preparation, event-driven interactions at the fair, and follow-up actions.
- Constituent domain actions (what the actor does):
  - Prepare/tailor résumé and portfolio.
  - Attend and check in.
  - Deliver elevator pitch at booths.
  - Hand over resume/portfolio.
  - Submit applications through portals and referrals.
  - Send follow-up messages (email/LinkedIn) or withdraw application.
- Constituent domain events (what happens in the domain):
  - Recruiting event announced.
  - Candidate checked in.
  - Recruiter solicited introduction.
  - Resume/portfolio handed over.
  - Application submitted through multiple portals.
- Expected outcomes: candidate visibility, duplicate records, and follow-up probability.

- Reference to sketch: Students prepare portfolios, deliver pitches, hand over resumes, and apply across multiple systems.

Fast screening and prioritization (pattern / behavior)

- Type: behavior / operational pattern
- Definition / justification: Recruiters manage volume by applying simple heuristics and rule-based filters — a pattern composed of repeated prescreen operations and evaluative events.
- Constituent actions:
  - Define heuristics (year, skills, seniority).
  - Apply prescreen operation to incoming records.
  - Flag or tag candidates in note-taking repository.
- Constituent events:
  - Pre-screen just completed.
  - Recruiter recorded notes.
- Risks/side effects: risk of overlooking strong profiles due to coarse filters.

Multichannel application handling and consolidation (behavior)

- Type: behavior — system + human coordination
- Definition / justification: A domain behavior involving repeated candidate submissions and system attempts to merge and deduplicate into a single source-of-truth.
- Constituent actions:
  - Candidate submits across portals.
  - System deduplicates and merges records.
  - Recruiter/system links records for traceability.
- Constituent events:
  - Application submitted through multiple portals.
  - Transfer logs updated.
- Expected outcomes: reduced duplication, but potential confusion if deduplication is imperfect.

Stopping after an early promising cohort (behavior / decision pattern)

- Type: behavior / decision pattern
- Definition / justification: A sourcing pattern where recruiters evaluate early candidates and then decide to pause or stop sourcing. This qualifies as a behavior because it combines evaluation actions and a sourcing-halting operation following recent evaluative events.
- Constituent actions:
  - Evaluate initial cohort of candidates (screening, interviews).
  - Make sourcing decision (operation).

    - Halt active sourcing operation.
- Constituent events:
    - Pre-screen just completed (for initial cohort).
    - Decision to stop sourcing was just made.
- Expected outcomes: narrowing of applicant window and reduced diversity of incoming profiles.
- Justification note: classifying this as a behavior makes explicit the action/event components rather than treating it as a single atomic "behavior."

Communication delay leading to attrition (behavior)

- Type: behavior
- Definition / justification: A time-sensitive pattern where delayed outreach produces candidate attrition; composed of scheduling actions and delayed-contact events.
- Constituent actions:
    - Queue candidates for later contact (operation).
    - Send invitations/follow-ups after delay.
- Constituent events:
    - Invitation or follow-up has been delayed.
    - Candidate withdraws or accepts another offer (resulting event).
- Expected outcomes: increased drop-off and lower conversion.

Informal networking conversion (behavior)

- Type: behavior
- Definition / justification: Sequence where unplanned interactions produce higher-quality leads than booth interactions.
- Constituent actions:
    - Informal conversation initiated by either party.
    - Focused follow-up scheduled and logged.
    - Interview scheduled or lead moved to pipeline.
- Constituent events:
    - Informal meetup occurred.
- Expected outcomes: stronger opportunities compared to rushed booth chats.

Notes as memory (pattern / entity usage)

- Type: pattern / entity usage
- Definition: Short tags and concise notes become the memory of many brief encounters; this is both a persistent-entity use (notes repository) and a recurring operation (note-taking).
- Constituent actions:

    - Write short tags (e.g., "has Python").
    - Link notes to contact records in repository.
- Constituent events:
    - Recruiter recording notes.

Rehiring based on prior trust (behavior)

- Type: behavior
- Definition / justification: Pattern where previous interns or employees are prioritized; it combines record-lookup actions with a rehire operation.
- Constituent actions:
    - Lookup past intern/employee records in system.
    - Prioritize outreach and offer.
- Constituent events:
    - Known talent has been rehired.
- Expected outcomes: faster hiring and reduced uncertainty.

Brand recall through giveaways (behavior / pattern)

- Type: behavior / marketing pattern
- Definition: Distribution of swag leads to higher brand recall and subsequent reengagement.
- Constituent actions:
    - Distribute promotional materials or swag.
    - Candidate later recalls brand and reengages during outreach.
- Constituent events:
    - Swag or promotional material was handed out.

2.1.6. Function Signatures

~~The system's core domain logic is defined by a set of function signatures that outline operations, inputs, outputs, and potential failure states. At its foundation, user interaction is governed by swiping and matching mechanics.~~

- ~~`getNextProfile : UserId -> Option ProfileCard` Fetches the next profile card from a user's personalized deck. Returns `None` if the deck is empty.~~
- ~~`processSwipe : UserId >< ProfileId >< SwipeDirection -> Result<Unit, Error>` Records a swipe (Like or Pass) for a given profile. Returns a result indicating success or an error.~~
- ~~`checkForMatch : UserId >< ProfileId -> Option Match` Determines whether a swipe action resulted in a mutual Like, returning a `Match` if successful.~~

~~Once a successful Match is established, profile and connection management functions handle communication setup:~~

- ~~`createConnection : Match → Connection` Creates a persistent connection between two matched users, establishing a channel for communication.~~
- ~~`sendMessage : ConnectionId >< UserId >< MessageContent → Result<Message, Error>` [line-through]#Sends a message within a connection. Returns the message if successful, or an error for invalid requests.~~
- ~~`getProfile : UserId → Result<Profile, Error>` Retrieves a full user profile (as opposed to the condensed profile card), with permission checks applied.~~

~~Session and state management functions ensure authentication and preparation of user data:~~

- ~~`initializeUserSession : UserCredentials → Result<UserSession, Error>` Authenticates a user's credentials. If successful, returns a session containing identity and role.~~
- ~~`getUserDeck : UserId → Deck` Builds a user's deck of profile cards, dynamically generated using the platform's matching algorithm and the user's past activity and preferences.~~

This section describes the key operations that define the system's behavior. Each function signature is presented with its conceptual explanation.

`Function: Future<ProfileCard?> getNextProfile(String userId)`

- This function fetches the next profile card from a user's personalized deck. The deck is dynamically generated based on if the user is a Candidate or Recruiter, their preferences, and active job openings. For a recruiter, this returns a `CandidateCard`; for a candidate, a JobCard. The function returns null when the current deck is exhausted, signaling there are no more profiles to review. This is a more abstract function.

`Function: Future<Result<void>> processSwipe(String userId, String profileId, SwipeDirection direction)`

- This operation records a user's initial evaluation of a profile. Conceptually, a "Like" corresponds to the domain action of judging a profile as Interested, while a "Pass" corresponds to NotInterested. This function serves as the UI trigger that would typically invoke the domain-level judge or dismiss functions in the backend, persisting the user's intent. It returns a Result type to handle potential errors, such as trying to swipe on an invalid profile.

`Function: Future<Match?> checkForMatch(String userId, String profileId)`

- Following a "Like" action, this function checks for mutual interest. It is the procedural counterpart to the declarative domain rule embodied in `checkForMutualInterest`. It queries the system state to determine if the profile the user just liked has also already liked them back. It returns a Match object upon this condition being met, otherwise null.

`Domain Function: judge(recruiter: Recruiter, candidate: Candidate, stage: CandidateStage) → Recruiter`

- This is the pure domain abstraction for the evaluation process. It models the recruiter's action of classifying a candidate into a specific evaluative state. The CandidateStage is a central domain type representing this progression: `CandidateStage = { Undecided, NotInterested, Interested, Invited, OfferMade }` The function takes the current state of the recruiter, the candidate in question, and the new stage, and returns an updated Recruiter entity.

`Domain Function: dismiss(recruiter: Recruiter, candidate: Candidate, position: JobOpening) → Recruiter`

- This function is a more specific form of judge, representing the domain action of a recruiter explicitly rejecting a candidate for a particular job opening. It is semantically clearer than `judge(…, NotInterested)` when the context of the specific position is important.

`Function: Future<Connection> createConnection(Match match)`

- This function is called after a Match is found. It creates a persistent Connection entity, which serves as the shared context for all future one-to-one interactions between the matched users. This entity houses the chat history and connection metadata, formally establishing the communication channel.

`Function: Future<Result<Message>> sendMessage(String connectionId, String userId, String content)`

- This operation allows a user to send a message within an established connection. It requires the specific connectionId to ensure the message is part of the correct conversation thread. It performs permissions checks and returns a Message entity on success, or an error for invalid requests.

`Domain Function: establishConnection(match: Match) → Connection`

- This is the conceptual model for the createConnection function. It represents the creation of a Connection as a direct consequence of a successful Match in the domain.

`Function: Future<Result<UserSession>> initializeUserSession(UserCredentials credentials)`

- This function authenticates a user based on their credentials. Upon success, it returns a UserSession entity containing the user's identity and role, which is necessary for authorizing subsequent operations. A failure result indicates invalid authentication.

`Function: Future<Result<Profile>> getFullProfile(String userId, String requesterId)`

- This retrieves a user's complete profile. Crucially, it incorporates a domain-level permission check: the requesterId parameter is used to enforce business rules about who is allowed to see which parts of a profile. This is a more nuanced and secure domain concept than a simple data fetch.

Scenario: A Recruiter Reviews a Candidate

1. Fetch Profile: The recruiter's app calls `getNextProfile("recruiter_123")`. The system returns a CandidateCard for candidate_abc.

2. Express Interest: The recruiter swipes right (Like). The app calls processSwipe("recruiter_123", "candidate_abc", SwipeDirection.like). Internally, this likely triggers a backend service that executes the domain action judge(recruiter_123, candidate_abc, Interested).

3. Check for Match: The system now calls `checkForMatch("recruiter_123", "candidate_abc")`. It discovers that candidate_abc had already "liked" a job opening from recruiter_123's company. Consequently, a Match object is returned.

4. Establish Connection: The frontend, upon receiving the Match, calls `createConnection(match)` to create a persistent Connection between the two parties, which is the conceptual establishConnection domain action.

5. Communication: The recruiter now uses `sendMessage("connection_xyz", "recruiter_123", "Welcome to our talent pool!")` to initiate contact within the sanctioned context of the new connection.

2.2. Requirements

~~User Stories, Epics, Features~~

~~This section outlines the experience our platform aims to provide for people seeking opportunities and for the organizations hiring them. We present the product in three layers short narrative user stories, the broader areas of value they belong to (epics), and the capabilities that bring those areas to life (features).~~

~~===== User stories~~

~~• *As a candidate, I want to create my profile with my background and skills so that recruiters can quickly assess my fit.*~~

~~**Acceptance criteria:**~~

~~**Given**~~	~~**When**~~	~~**Then**~~
~~I'm signed in~~	~~I complete required fields and save~~	~~My profile is published~~
~~Required info is missing~~	~~I try to publish~~	~~I'm shown exactly which fields to complete~~

~~• *As a candidate, I want to add portfolio items (PDF or public URLs) so that my work is easy to review.*~~

~~**Acceptance criteria:**~~

~~**Given**~~	~~**When**~~	~~**Then**~~
~~A valid file/URL~~	~~I upload~~	~~The item appears in a gallery I can reorder~~
~~An unsupported type/size~~	~~I upload~~	~~I see an error listing allowed types and max size~~

~~• *As a candidate, I want to choose my profile visibility (public / by Match / private) so that I control my exposure.*~~

~~**Acceptance criteria:**~~

~~**Given**~~	~~**When**~~	~~**Then**~~
~~"Private" visibility~~	~~Recruiters search~~	~~My profile does not appear in results~~
~~"By Match" visibility~~	~~A mutual Match occurs~~	~~My profile becomes visible to the matched party~~

~~Given~~	~~When~~	~~Then~~
~~"Public" visibility~~	~~Recruiters search~~	~~My profile can appear in recruiter results~~

~~• *As a recruiter, I want to publish a simple company page so that candidates understand who we are and our roles.*~~

~~**Acceptance criteria:**~~

~~Given~~	~~When~~	~~Then~~
~~Logo, short description, sectors, and location are provided~~	~~I publish~~	~~The company page is visible to candidates~~
~~Required info is incomplete~~	~~I attempt to publish~~	~~I'm prompted to complete the missing fields~~

~~• *As a recruiter, I want to filter candidates by skills, role interest and availability so that I can shortlist relevant profiles.*~~

~~**Acceptance criteria:**~~

~~Given~~	~~When~~	~~Then~~
~~Combined filters~~	~~I search~~	~~Results highlight matched terms~~
~~Sample data under normal conditions~~	~~I search~~	Results load in ~2 seconds (p95)#

~~• *As a candidate, I want to express interest with a quick like or pass so that I can move fast through options.*~~

~~**Acceptance criteria:**~~

~~Given~~	~~When~~	~~Then~~
~~A profile is shown~~	~~I press Like~~	~~My interest is stored~~
~~A profile is shown~~	~~I press Pass~~	~~That profile does not appear again in the current session~~

~~• _As a user (candidate or recruiter), I want to be notified when there's a mutual like so that we can start a conversation.~~

Acceptance criteria:

~~Given~~	~~When~~	~~Then~~
~~Both sides liked each other~~	~~The system detects a mutual Like~~	~~A chat thread opens and an in-app notification is sent~~

~~• *As a matched user, I want to exchange messages so that we can coordinate next steps.*~~

~~**Acceptance criteria:**~~

Given	When	Then
A Match chat is open	I send a message	The recipient receives it near real-time and I see sent/read states
I have blocked the other party	They try to message me	The message is not sent and I receive no notification

• *As a user, I want to report or block a profile so that I feel safe using the platform.*

Acceptance criteria:

Given	When	Then
I submit a report	-	A moderation case is created for review
I block a profile	-	It no longer appears in my experience and cannot open new chats

• *As a candidate, I want to see upcoming recruiting events and RSVP so that I don't miss opportunities.*

Acceptance criteria:

Given	When	Then
I RSVP to an event	The event is 24h away	An in-app reminder is delivered
The events feed is available	-	Items are ordered by date and show title, location, and registration/RSVP

===== Epics - Candidate Profile and Portfolio - Recruiter Discovery and Search - Matching and Messaging - Events and Notifications - Safety and Moderation

2.2.1. User Stories, Epics, Features

NOTE	This subsection defines the product scope from a user-value perspective. It organizes the solution into Epics that capture high-level goals and Features that realize those goals in the system.

Abbreviations and ID Conventions:

Abbrev.	Meaning
US	User Story: a user-centered need framed as intent and value.
E	Epic: a high-level goal that groups related features and stories.
F	Feature: a concrete capability that realizes part of an epic.
ReqRef	Requirement Reference: the requirement ID(s) a story or feature maps to.
REQ	Requirement: a functional or non-functional specification with a stable ID.

Identifier Formats:

Type	Format	Components	Example
User Story ID:	`US.AREA.NN`	`US` = user story.; `AREA` = functional area (e.g., `PROF`, `SRCH`, `MATCH`, `CHAT`, `EVT`, `NOTIF`, `SAFE`); `NN` = two-digit sequence.	`US.PROF.01`
Epic ID:	`E#` or "Epic E#: Title".	`#` = epic number.; clear title preferred in headings.	`E1` or `Epic E1: Candidate Profile & Portfolio`
Feature ID:	`F#.N`	`#` = epic number; `N` = feature sequence within that epic.	`F1.1`, `F3.2`
Requirement ID:	`REQ-AREA-TYPE-NN`	`AREA` = functional area; `TYPE` = `F` (Functional) or `NF` (Non-functional); `NN` = two-digit sequence.	`REQ-PRF-F-01`, `REQ-CHAT-NF-01`

ReqRef Usage (inside a story or feature):

Placement	Syntax
Same line as the title:	`==== US.PROF.01: Publish a complete profile ReqRef: REQ-PRF-F-01.`
Next line below the title:	`==== US.PROF.01: Publish a complete profile` `ReqRef: REQ-PRF-F-01.`

Epic E1: Candidate Profile & Portfolio

Goal: Present credible competence fast.

Problem/value: Candidates need a concise, verifiable profile that allows recruiters to assess fit within seconds.

Features (F1):

- F1.1 Profile editor: The profile editor captures a candidate's education, skills, experience, and role interests.
- F1.2 Portfolio artifacts: Candidates can upload portfolio items such as PDFs, public links, and videos, and they can reorder those items.
- F1.3 Visibility and privacy controls: Candidates can set their profile visibility to Public, Match-only, or Private and retain full control over exposure.
- F1.4 Profile completeness indicator: The system displays a completeness indicator that shows progress toward a fully publishable profile.

US.PROF.01: Publish a complete profile | ReqRef: REQ-PRF-F-01

"As a candidate, I want to publish my education, skills, and experience so that recruiters can evaluate fit quickly."

Assumptions/Dependencies: verified university email; identity/enrollment verification available.

Acceptance criteria:

Given	When	Then
A verified account.	All required fields are completed and saved.	The profile is published and listed as “Complete”.
Required info is missing.	Publish is attempted.	Inline errors show exactly which fields remain.
Profile is updated.	Changes are saved.	The last-updated timestamp is refreshed.

US.PROF.02: Add portfolio items | ReqRef: REQ-PRF-F-02

"As a candidate, I want to add portfolio items so that my work is easy to review."

Acceptance criteria:

Given	When	Then
A valid file or public URL	The item is uploaded.	The item appears in a gallery and can be reordered.
An unsupported type or size.	Upload is attempted.	An error lists allowed types and maximum size.

US.PROF.03: Control profile visibility | ReqRef: REQ-PRF-F-03

"As a candidate, I want to choose my profile visibility so that I control my exposure."

Acceptance criteria:

Given	When	Then
Visibility is set to “Private”.	Recruiters search.	The profile does not appear in results.
Visibility is set to “Match-only”.	A mutual Match occurs.	The profile becomes visible to the matched party.
Visibility is set to “Public”.	Recruiters search.	The profile can appear in results.

E1 Traceability (Stories → Features → Requirements):

Story ID	Feature	ReqRef	Notes
US-PROF-01	F1.1, F1.4	REQ-PRF-F-01	Completeness logic and publish rules.

Story ID	Feature	ReqRef	Notes
US-PROF-02	F1.2	REQ-PRF-F-02	File/URL validation constraints.
US-PROF-03	F1.3	REQ-PRF-F-03	Access control and search filtering.

Epic E2: Recruiter Discovery & Search

Goal: Shortlist qualified candidates efficiently.

Problem/value: Recruiters need to discover relevant candidates quickly and understand organizational context without friction.

Features (F2):

- F2.1 Company page: The organization can publish a company page with logo, sectors, locations, and available roles.
- F2.2 Candidate search with filters: Recruiters can search using filters such as skills, role interests, and availability.
- F2.3 Candidate detail view: Recruiters can open a detailed candidate view that consolidates profile and portfolio information.

US.SRCH.01: Publish a company page | ReqRef: REQ-SRCH-F-01

"As a recruiter, I want to publish a simple company page so that candidates understand who we are and our roles."

Acceptance criteria:

Given	When	Then
Logo, description, sectors, and location are provided.	Publishing is requested.	The page becomes visible to candidates.
Required info is incomplete.	Publishing is requested.	Prompts indicate missing fields.

US.SRCH.02: Filter and rank candidates | ReqRef: REQ-SRCH-F-02

"As a recruiter, I want to filter candidates by skills, role interest, and availability so that I can shortlist relevant profiles."

Acceptance criteria:

Given	When	Then
Combined filters.	Search is executed.	Results highlight matched terms.
Normal traffic and sample data.	Search is executed.	Results load in ~2 seconds (p95).

E2 Traceability (Stories → Features → Requirements):

Story ID	Feature	ReqRef	Notes
US-SRCH-01	F2.1	REQ-SRCH-F-01	Company profile schema.
US-SRCH-02	F2.2, F2.3	REQ-SRCH-F-02	Filter set and performance target.

Epic E3: Matching & Messaging

Goal: Move from interest to conversation quickly.

Problem/value: Both parties need a fast way to express interest, form a mutual Match, and start secure conversations.

Features (F3):

- F3.1 Like and Pass interactions: Users can register quick likes or passes on presented profiles.
- F3.2 Mutual Match and notification: The system detects mutual interest and triggers an in-app notification that opens a chat.
- F3.3 One-to-one chat: Matched users can exchange messages with delivery and read states.

US.MATCH.01: Express quick interest | ReqRef: REQ-MATCH-F-01

"As a candidate, I want to like or pass quickly so that I can move fast through options."

Acceptance criteria:

Given	When	Then
A profile is shown.	Like is pressed.	Interest is stored.
A profile is shown.	Pass is pressed.	That profile is removed from the current session.

US.MATCH.02: Get notified on mutual like | ReqRef: REQ-MATCH-F-02

"As a user, I want to be notified when there is a mutual like so that a conversation can start."

Acceptance criteria:

Given	When	Then
Both sides liked each other.	The system detects mutual like.	A chat thread opens and an in-app notification is sent.

US.CHAT.01: Exchange messages with safety | ReqRef: REQ-CHAT-F-01

"As a matched user, I want to exchange messages so that next steps can be coordinated."

Acceptance criteria:

Given	When	Then
A Match chat is open.	A message is sent.	The recipient receives it near real time; the sender sees sent and read states.
The other party is blocked.	They attempt to send a message.	The message is not delivered and no notification is generated.

E3 Traceability (Stories → Features → Requirements):

Story ID	Feature	ReqRef	Notes
US-MATCH-01	F3.1	REQ-MATCH-F-01	Interaction logging.
US-MATCH-02	F3.2	REQ-MATCH-F-02	Match detection and notification trigger.
US-CHAT-01	F3.3	REQ-CHAT-F-01	Realtime delivery, receipts, block rules.

Epic E4: Events & Notifications

Goal: Increase attendance and timely follow-through.

Problem/value: Candidates must discover opportunities in time and receive reminders that respect preferences and quiet hours.

Features (F4):

- F4.1 Events feed: The system lists events with title, date and time, location, and RSVP state.
- F4.2 RSVP and reminders: Users can RSVP and receive reminders before the event.
- F4.3 Notification preferences: Users can configure quiet hours and choose preferred channels.

US.EVT.01: Discover and RSVP to events | ReqRef: REQ-EVT-F-01

"As a candidate, I want to see upcoming recruiting events and RSVP so that opportunities are not missed."

Acceptance criteria:

Given	When	Then
Events feed is available.	-	Items are ordered by date and show title, location, and RSVP.

Given	When	Then
An RSVP exists.	The event is 24 hours away.	An in-app reminder is delivered.

US.NOTIF.01: Respect notification preferences | ReqRef: REQ-NOTIF-F-01

"As a user, I want notifications to follow my channel and quiet-hour settings so that interruptions are minimized."

Acceptance criteria:

Given	When	Then
Quiet hours are active.	A non-urgent event occurs.	Notifications are queued until quiet hours end.
The user opted in to in-app only.	A reminder must be sent.	Only in-app is used; no email or SMS is sent.

E4 Traceability (Stories → Features → Requirements):

Story ID	Feature	ReqRef	Notes
US-EVT-01	F4.1, F4.2	REQ-EVT-F-01	RSVP state and reminders.
US-NOTIF-01	F4.3	REQ-NOTIF-F-01	Quiet hours and channel policy.

Epic E5: Safety & Moderation

Goal: Maintain a safe, trusted environment.

Problem/value: Users must be able to report issues and block unwanted contacts, and moderators need clear workflows.

Features (F5):

- F5.1 Report a profile: Users can submit a report for moderation review.
- F5.2 Block or unblock a user: Users can block or later restore interaction with another profile.
- F5.3 Moderation review queue: Administrators can triage and process reported cases.

US.SAFE.01: Report inappropriate behavior | ReqRef: REQ-SAFE-F-01

"As a user, I want to report a profile so that moderation can review and act."

Acceptance criteria:

Given	When	Then
A report is submitted.	-	A moderation case is created with timestamp and reporter ID.

US.SAFE.02: Block interactions | ReqRef: REQ-SAFE-F-02

"As a user, I want to block a profile so that it no longer appears or can initiate chats."

Acceptance criteria:

Given	When	Then
Block action is confirmed.	-	The blocked profile no longer appears and new chats cannot be opened.
Unblock is requested.	-	Visibility and messaging return to the pre-block state.

E5 Traceability (Stories → Features → Requirements):

Story ID	Feature	ReqRef	Notes
US-SAFE-01	F5.1, F5.3	REQ-SAFE-F-01	Moderator workflow.
US-SAFE-02	F5.2	REQ-SAFE-F-02	Block and unblock policy with propagation.

2.2.2. Personas

The personas below represent our core user segments and ground the scope of this product. For each persona we outline goals, pains, typical behaviors, and accessibility needs, and we link them to the stories, epics, and features defined in 2.2.1. We'll reference these personas by name during planning and reviews to keep decisions concrete and tied to user value.

Table 1. María "New Grad" Rivera — University Candidate (mobile-first)

Snapshot	Loves hackathon weekends and cafés near campus; anxious about first-job search but optimistic.
Background	22, UPRM (CS). First-gen grad, part-time tutoring; lives off-campus with roommates.
Motivations	Land her first SWE role where she can keep learning; wants fast, clear signals of interest.
Hobbies	Campus hackathons, short video reels of projects, weekend hikes.
Tech Setup	iPhone as primary device; edits portfolio on a shared laptop.

Relationship to App	Wants quick Like/Pass and reminders for events tied to companies she follows.
Goals	Publish a complete profile quickly; showcase a simple portfolio; control visibility; get event reminders.
Pains	Long forms; vague errors; unwanted exposure.
Behavior	Short sessions; prefers simple actions (Like/Pass).
Accessibility	Clear, actionable error messages; low latency on mobile.
Related Stories	Create profile; Add portfolio; Choose visibility; Like/Pass; Match notification; 1:1 messaging; Events feed & RSVP.
Epics	Candidate Profile & Portfolio; Matching & Messaging; Events & Notifications; Safety & Moderation.
Quote	"I want to upload the essentials and start exploring without oversharing."

Table 2. Luis “Switcher” Santiago — Career-transition Candidate (privacy-first)

Snapshot	Careful planner changing lanes into QA; values control and signal quality over volume.
Background	30, IT support → moving into QA; evening bootcamp; helping family on weekends.
Motivations	Show real, verifiable skills without broadcasting a job search to current contacts.
Hobbies	Keyboard mods, bug-bash meetups, journaling progress.
Tech Setup	Desktop first; tracks opportunities in spreadsheets.
Relationship to App	Prefers “visibility by Match”; wants strong filters and concise profile previews.
Goals	Import/organize history; highlight skills; appear in relevant searches without going fully public.
Pains	Lack of control over who sees his profile; imprecise recruiter filters.
Behavior	Logs in a few times per week; replies only when there’s a real Match.
Accessibility	Desktop-oriented; concise summaries.
Related Stories	Choose visibility (private/by-Match/public); Profile & portfolio; 1:1 messaging.
Epics	Candidate Profile & Portfolio; Matching & Messaging; Safety & Moderation.
Quote	"I want to be visible only to people who truly match with me."

Table 3. Karla “Campus Recruiter” Gómez — Recruiter (events & funnel)

Snapshot	Organized, metric-driven; splits time between campus events and fast triage.

Background	Tech company recruiter; owns 3 junior openings; coordinates campus tours with a small team.
Motivations	Build a clean funnel quickly; reduce no-shows; capture reliable signals pre-event.
Hobbies	Morning runs; mentors student groups; podcast commutes.
Tech Setup	Laptop + ATS tabs; lives in filters and saved searches.
Relationship to App	Needs crisp company page, combined filters, and event RSVP with reminders.
Goals	Publish company page; filter by skills/interest/availability; view candidate detail; manage RSVPs and reminders.
Pains	Noisy results; search latency; incomplete candidate info.
Behavior	1–2 h desktop sessions; heavy use of combined filters; saves shortlists.
Service Level	Search with sample data should load in ~2s (p95).
Related Stories	Company page; Search with filters; Results highlight matched terms; Events feed & RSVP; Notifications.
Epics	Recruiter Discovery & Search; Events & Notifications.
Quote	"I need ten viable profiles in minutes and a way to nurture them to interview."

Table 4. Jorge “HR Generalist” Ortiz — SMB Recruiter (speed & safety)

Snapshot	Wears many hats; wants quick, safe conversations that don’t waste cycles.
Background	HR at a 35-person firm; manages onboarding, payroll, and recruiting.
Motivations	Shortlists fast; protect team time; keep the conversation professional and safe.
Hobbies	Weekend leagues; DIY home projects.
Tech Setup	Older office desktop; checks mobile during site visits.
Relationship to App	Needs practical filters, read receipts, and easy report/block.
Goals	Filter by skills and availability; chat 1:1; report or block bad behavior.
Pains	Incomplete profiles; spam/inappropriate contacts.
Behavior	Short work blocks; values online indicators and read receipts.
Related Stories	Filter by skills/interest/availability; 1:1 chat with sent/read states; Report/Block.
Epics	Recruiter Discovery & Search; Matching & Messaging; Safety & Moderation.
Quote	"Give me a reliable shortlist and a clear conversation; the rest is noise."

Table 5. Ana “Safe User” Lozada - Safety-focused Candidate (safety-first)

Snapshot	Cautious first-timer; wants control and predictable notifications.

Background	24, first time on a jobs platform; previous negative social app experiences.
Motivations	Try a new channel without risking privacy or overwhelm.
Hobbies	Photography walks, language exchange groups.
Tech Setup	Android mid-range; limits notifications outside 9–6.
Relationship to App	Wants visibility controls, block/report, and meaningful alerts only.
Goals	Block or report profiles; prevent re-appearance after Pass; receive only useful notifications.
Pains	Unwanted interactions; intrusive alerts.
Behavior	Reviews privacy settings; uses reporting if something feels unsafe.
Accessibility	Simple controls for privacy, block, and report.
Related Stories	Report/Block; Like/Pass does not re-show in session; Relevant in-app notifications.
Epics	Safety & Moderation; Matching & Messaging; Events & Notifications.
Quote	"I want to feel in control and safe at all times."

Table 6. Mina “International Grad” Shah - International Candidate (compliance-first)

Snapshot	International MS grad navigating visas and time zones; needs clarity and eligibility signals.
Profile	24, MS in Data Science, international student; lives off-campus; phone for browsing, laptop for uploads.
Goals	Visa-friendly profile & portfolio; appear in searches filtered by skills, location, and authorization; timely Match notifications; RSVP and reminders; safe messaging.
Pains	Ambiguous job location and start date; unclear offer validity; outreach without consent; slow search; duplicate event notices.
Behavior	Curates projects weekly; short-burst swipes; evening chats; shortlists companies; RSVPs to virtual events.
Accessibility	Clear copy on compensation (salary + benefits), readable tables, timezone-aware reminders.
Related Stories	Create profile & portfolio; Choose visibility; Recruiter search (skills, location, authorization, availability); Match notification; 1:1 messaging; Events feed & RSVP; Report/Block.
Epics	Candidate Profile & Portfolio; Recruiter Discovery & Search; Matching & Messaging; Events & Notifications; Safety & Moderation.
Quote	"Make it crystal clear where the role is, whether I’m eligible, and ping me when it’s a real match—then I can move fast."

Coverage matrix (personas × epics)

Persona	Candidate Profile & Portfolio	Recruiter Discovery & Search	Matching & Messaging	Events & Notifications	Safety & Moderation
María (New Grad)	X		X	X	X
Luis (Switcher)	X		X		X
Karla (Recruiter)		X		X	
Jorge (HR Gen.)		X	X		X
Ana (Safe User)			X	X	X
Mina (Intl. Grad)	X	X	X	X	X

2.2.3. Domain Requirements

The domain requirements are the fundamental rules and constraints of the problem space, independent of the software implementation.

ID	Description	Linked User Stories/Epics	Verification Tests
DR001	The system must uniquely identify every candidate, recruiter, employer, and event organizer, distinguishing natural persons from institutional profiles.	US-Auth-001: Register as Candidate, US-Auth-002: Register as Recruiter	T-DB-001, T-API-001
DR002	The system must record the role performed by each entity in every interaction, along with the time and context of that role.	US-Interaction-001: Express Interest, US-Interaction-002: Create Match	T-Log-001, T-DB-002
DR003	The system must represent every opening with explicit requirements (min. experience, qualifications, certifications, languages, legal authorizations, work modality, location, compensation range, and start date).	Epic-JobMgmt	T-API-002, T-UI-001
DR004	The system must record the number of positions available for each opening and link every opening to its responsible employer.	US-Recruiter-001: Post New Opening	T-DB-003, T-API-003

ID	Description	Linked User Stories/Epics	Verification Tests
DR005	The system must allow requirement updates for an opening while preserving a complete history of changes.	US-Recruiter-002: Update Opening	T-Audit-001
DR006	The system must represent each candidate profile with education, work history, competencies, certifications, portfolios, and verifiable references.	Epic-ProfileMgmt	T-DB-004
DR007	The system must allow attaching documents to the profile, preserving issue date, validity, and verification status.	US-Candidate-001: Upload Resume	T-File-001
DR008	The system must record candidate preferences such as areas of interest, desired location, and work modality.	US-Candidate-002: Set Preferences	T-DB-005
DR009	The system must represent recruiting events with name, organizer, venue, agenda, date, and capacity, and represent company booths/tables.	Epic-EventMgmt	T-DB-006, T-API-004
DR010	The system must register attendance and arrival of candidates and recruiters, and enforce registration rules and capacity limits.	US-Event-001: Check-in at Event	T-Event-001, T-API-005
DR011	The system must model queues in front of booths/tables preserving a first-come, first-served order.	Epic-QueueMgmt	T-Queue-001
DR012	The system must allow limiting the duration of each turn, register the closure of each turn, and prevent assigning turns that exceed declared capacity.	US-Event-002: Manage Queue	T-Queue-002
DR013	The system must allow candidates and recruiters to express interest in openings or profiles and record reciprocal expressions.	US-Interaction-001: Express Interest	T-API-006
DR014	The system must create a Match when both sides express positive interest, preserving the date and context of that event.	US-Interaction-002: Create Match	T-Match-001
DR015	The system must allow recruiters to define shortlisting criteria and register the decision with its justification.	US-Recruiter-003: Shortlist Candidate	T-API-007
DR016	The system must allow candidates to withdraw their interest and update any pending Matches.	US-Candidate-003: Withdraw Interest	T-Match-002

ID	Description	Linked User Stories/Epics	Verification Tests
DR017	The system must represent availability of candidates and recruiters through calendars and time blocks.	Epic-Scheduling	T-DB-007
DR018	The system must schedule interviews only on valid Matches, prevent double booking, and register interview outcomes with clear states (continues, rejected, offer extended) and date/responsible party.	US-Scheduling-001: Book Interview	T-Schedule-001
DR019	The system must allow message exchanges between candidate and recruiter only when there is a valid Match or explicit candidate permission.	US-Communication-001: Send Message	T-Comm-001
DR020	The system must register candidate consent for sharing information with an employer or organizer and allow revocation of that consent.	US-Candidate-004: Manage Consent	T-DB-008
DR021	The system must compute and display the degree of requirement fulfillment for each candidate (meets, partially meets, does not meet).	US-Recruiter-004: View Match Score	T-Algo-001
DR022	The system must prevent practices that alter queue order without a rule defined by the event (e.g., priority for confirmed appointments).	T-Queue-003, T-Security-001	DR023
The system must keep an auditable record of shortlisting and rejection decisions, together with their criteria.	T-Audit-002	DR024	The system must notify candidates and recruiters when a Match is created, an interview is confirmed, or when changes to opening requirements affect eligibility.

ID	Description	Linked User Stories/Epics	Verification Tests
US-Notification-001	T-Notif-001	DR025	The system must notify candidates about recruiting events related to employers or openings in which they have shown interest.
US-Notification-002	T-Notif-002	DR026	The system must record domain metrics (Match rate, queue waiting time, average turn duration, follow-up rate, event attendance) and support funnel analysis from expression of interest to offer extended.
Epic-Reporting	T-Metric-001	DR027	The system must prevent the creation of Matches when any party does not exist or when the opening is closed.
T-API-008, T-Security-002	DR028	The system must keep the link between every document and the identity of the person/entity that provided it, together with its validation status.	T-File-002

~~The system must uniquely identify every candidate, every recruiter, every employer, and every event organizer, while distinguishing natural persons from institutional profiles when the entity is a company or an organizer, and it must record the role performed by each entity in every interaction together with the time and the context of that role. The system must represent every opening with explicit requirements including minimum experience, qualifications, certifications, languages, legal authorizations, work modality, location, compensation range, and start date, must record the number of positions available for each opening, must link every opening to its~~

~~responsible employer, and must allow requirement updates while preserving a complete history of changes. The system must represent each candidate profile with education, work history, competencies, certifications, portfolios, and verifiable references, must allow attaching documents to the profile preserving issue date, validity, and verification status, and must record candidate preferences such as areas of interest, desired location, and work modality. The system must represent recruiting events with name, organizer, venue, agenda, date, and capacity, must represent company booths or tables within each event and their relation to the promoted openings, must register attendance and arrival of candidates and recruiters, and must enforce registration rules and capacity limits defined by the organizer. The system must model queues in front of booths or tables preserving a first come first served order, must allow limiting the duration of each turn, must register the closure of each turn by staff, and must prevent assigning turns that exceed the declared capacity of a booth or of a time block. The system must allow candidates and recruiters to express interest in openings or in profiles and must record when the expression is reciprocal, must create a Match when both sides express positive interest and must preserve the date and the context of that event, must allow recruiters to define shortlisting criteria based on requirements and evidence and must register the decision with its justification, and must allow candidates to withdraw their interest and update any pending Matches. The system must represent availability of candidates and recruiters through calendars and time blocks, must schedule interviews only on valid Matches, must prevent double booking of the same block, and must register interview outcomes with clear states such as continues in process, rejected, or offer extended together with the date and the responsible party. The system must allow message exchanges between candidate and recruiter only when there is a valid Match or an explicit permission from the candidate, must register candidate consent for sharing information with an employer or with an organizer, and must allow revocation of that consent. The system must compute and display the degree of requirement fulfillment for each candidate with categories such as meets, partially meets, or does not meet, must prevent practices that alter queue order without a rule defined by the event such as priority for confirmed appointments, and must keep an auditable record of shortlisting and rejection decisions together with their criteria. The system must notify candidates and recruiters when a Match is created, when an interview is confirmed, and when changes to opening requirements affect eligibility, and it must notify candidates about recruiting events related to employers or openings in which they have shown interest. The system must record domain metrics such as Match rate, queue waiting time, average turn duration, follow up rate after Matches, and event attendance, and it must support funnel analysis from expression of interest to offer extended in order to identify bottlenecks. The system must prevent the creation of Matches when any party does not exist or when the opening is closed, must keep the link between every document and the identity of the person or the entity that provided it together with its validation status, and must preserve the full state history of each opening and of each candidacy from origin to closure.~~

2.2.4. Interface Requirements

Interface requirements specify the user-facing aspects of the system.

ID	Description	Linked User Stories/Epics	Verification Tests
IR001	The system must provide a structured form during registration for new recruiters to initialize their profile (capturing company name, professional email, and job role).	US-Auth-002: Register as Recruiter	T-UI-010, T-FE-001
IR002	The initialization process must include a validation step where a confirmation email is sent, and the internal profile must not be active until validation is complete.	T-BE-001, T-Email-001	IR003
The system must provide an edit profile screen for students to update their skills/competencies (e.g., adding new projects).	US-Candidate-005: Update Skills	T-UI-011	IR004
The internal skill representation must be updated immediately upon the student saving the changes.	T-BE-002	IR005	The system must present a visual notification to both users immediately upon a Match being recorded.

ID	Description	Linked User Stories/Epics	Verification Tests
US-Notification-001	T-UI-012, T-Notif-003	IR006	The Match notification interface must clearly indicate who the Match is with and include a prominent means to initiate communication.
T-UI-013	IR007	The system must provide a company profile management screen for authorized recruiters to correct company information (e.g., description, website URL).	US-Recruiter-005: Update Company Info
T-UI-014	IR008	The system must log all changes made via the company profile management screen.	T-Log-002
IR009	The company name field shall be immutable through the company profile management screen after initial validation; changes must be handled by a separate administrative process.	T-FE-002, T-Security-003	IR010

~~The first requirement is that the system must provide a means for a new user identifying as a recruiter to initialize their internal recruiter profile representation. This initialization shall be performed by the recruiter via a structured form presented during the registration process. The form must include fields to capture the shared phenomena of company name, professional email address, and job role. The initialisation process shall include a validation step where a confirmation email is sent to the provided professional email address. The internal profile representation shall not be considered active until this validation is complete. Secondly, the system must provide a means to update the internal representation of a student's skills when these phenomena change in the domain such as adding new projects. The student shall be able to initiate an update via an edit profile screen. The system shall provide input fields and controls to allow the student to modify their skill set. The internal representation shall be updated immediately upon the student saving the changes. The next requirement is that the internal event of a Match must be communicated to the involved users to reflect this new shared state in the domain. The system shall present a visual notification to both users immediately upon the Match being recorded in the system. This interface must provide a clear indication of who the Match is with and shall include a prominent means to initiate communication, thereby enabling the next domain action. The system must provide a~~

~~means for an authorized recruiter to correct the internal representation of their company's information if it was initially entered incorrectly or becomes outdated. Recruiters shall have access to a company profile management screen. This interface shall allow them to edit fields such as company description and website URL. The system shall log all such changes. The company name field shall be immutable through this interface after the initial validation to ensure traceability. Changes to the company name must be handled by a separate administrative process to maintain data integrity. Lastly, the internal system state must be updated to reflect a user's swipe action on a profile card. The user's gesture shall be the sole initiating action. The system shall immediately update the internal state to record this decision and remove the presented profile card from the user's current deck. No explicit save or confirm action is required; the gesture itself is the interface event.~~

2.2.5. Machine Requirements

Machine requirements specify the hardware, software, and environment needed for development, testing, and deployment.

ID	Description	Linked User Stories/Epics	Verification Tests
MR001	Developer Workstations must meet a minimum specification of Dual-core processor (Intel i5/AMD equivalent), 8 GB RAM, and 256 GB SSD.	Epic-DevSetup	T-Env-001
MR002	Mobile Devices for Testing must include Android devices running Android 10.0+ with at least 3 GB RAM and iOS devices running iOS 15+ (e.g., iPhone 11 or newer).	Epic-Testing	T-Env-002
MR003	Servers / Cloud Hosting must meet a minimum specification of 2 vCPUs, 4 GB RAM, and 50 GB storage.	Epic-Deployment	T-Env-003
MR004	The Flutter SDK (latest stable release) and Dart SDK (bundled with Flutter) must be used for development.	T-Tool-001	MR005
Android Studio and Xcode must be used for builds and emulators/simulators.	T-Tool-002	MR006	Git and GitHub must be used for version control and collaboration.

ID	Description	Linked User Stories/Epics	Verification Tests
T-Tool-003	MR007	SQLite must be used for local offline storage.	T-DB-009
MR008	Firebase, AWS, or Azure must be used for authentication, notifications, and backend integration.	T-API-009	MR009
Developer machines must run Windows 10/11 or macOS Monterey +.	T-Env-004	MR010	The application must be deployed to targets running Android 10+ and iOS 15+.
T-Env-005	MR011	A stable broadband connection (≥10 Mbps) must be available for syncing and testing cloud services.	T-Env-006
MR012	All client-server communication must use HTTPS.	T-Security-004	MR013
Developers must test on both Android and iOS environments.	T-Test-001	MR014	The application must comply with Google Play Store and Apple App Store distribution guidelines.

~~This section specifies the hardware, software, and environmental requirements to develop, test, and deploy the Professional Portfolio application. These requirements are tailored to the technologies chosen for the project: Flutter/Dart for the mobile frontend, cloud services (Firebase, AWS, or Azure) for backend support, and SQLite for local storage. **Hardware Requirements Developer Workstations** - Minimum: Dual-core processor (Intel i5/AMD equivalent), 8 GB RAM, 256 GB SSD. - Recommended: Quad-core processor, 16 GB RAM, 512 GB SSD. **Mobile Devices for Testing** - Android: Devices running Android 10.0+ with at least 3 GB RAM. - iOS: Devices running iOS 15+ (e.g., iPhone 11 or newer). **Servers / Cloud Hosting** - Minimum: 2 vCPUs, 4 GB RAM, 50 GB storage. - Recommended: 4 vCPUs, 8 GB RAM, 100 GB storage with autoscaling enabled on cloud platforms. **Software Requirements Development Tools** - Flutter SDK (latest stable release). - Dart SDK (bundled with Flutter). - Android Studio for Android builds and emulators. - Xcode for iOS~~

builds and testing. - Git and GitHub for version control and collaboration. **Programming Environment** - Dart as the primary programming language. - Flutter framework for cross-platform UI and logic. - Emulator/simulator tools for Android and iOS. **Database and Services** - SQLite for local offline storage. - Firebase, AWS, or Azure for authentication, notifications, and backend integration. **Environmental Requirements Operating Systems** - Developer machines: Windows 10/11 or macOS Monterey+. - Deployment targets: Android 10+ and iOS 15+. **Network** - Stable broadband connection (≥10 Mbps) for syncing repositories, package downloads, and testing cloud-based services. - All client-server communication must use HTTPS. **Other Constraints** - Developers must test on both Android and iOS environments. - Compliance with Google Play Store and Apple App Store distribution guidelines. - External dependencies must be compatible with the current Flutter LTS release. **Rationale and Validation** The requirements ensure that all contributors can work effectively while keeping compatibility with the targeted platforms. Minimum specifications guarantee accessibility across the team, while recommended specifications support smoother emulator performance and integration with cloud services. Each requirement is measurable (e.g., OS versions, RAM, network speed) and traceable to the technologies already adopted by the project.

2.3. Implementation

2.3.1. Functionality Flow Diagram

Before diving into the implementation fragments, it is useful to visualize the overall functionality flow of the Professional Portfolio system. The following diagram shows how the main user interactions—such as authentication, profile management, swiping, and messaging—trigger backend processes and data persistence operations.

This visualization complements the subsequent implementation fragments by illustrating how user-facing features are realized through coordinated service logic and database operations, ensuring alignment between system behavior and architectural design.

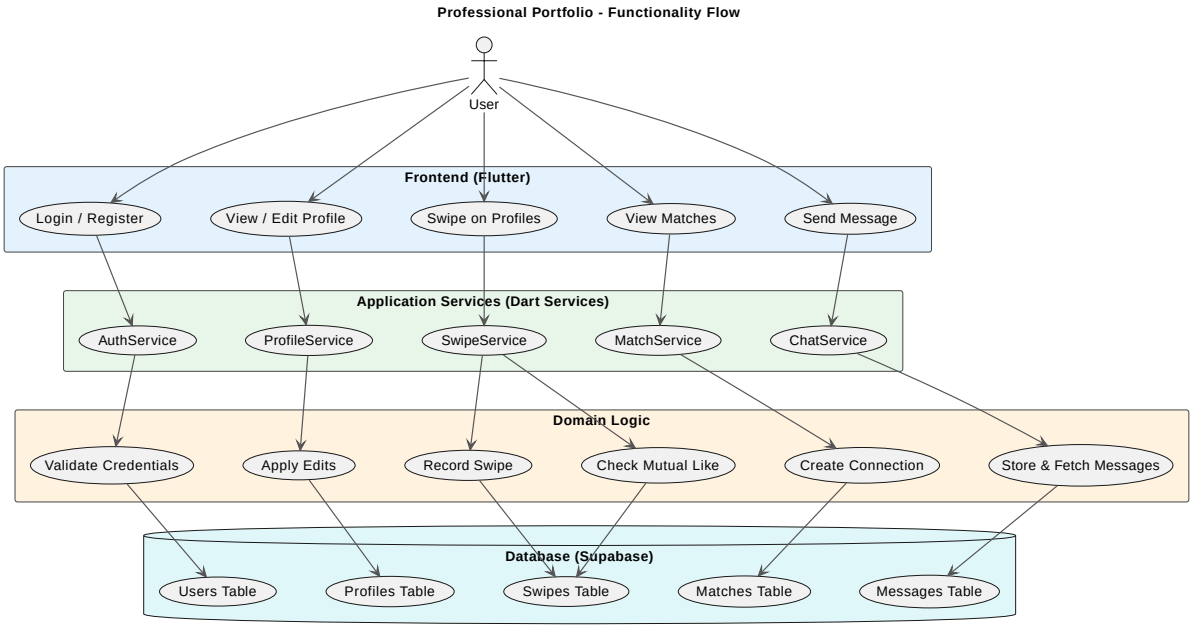


Figure 7. Functionality Flow Diagram illustrating the end-to-end process from user interactions to backend persistence.

The diagram clarifies how user interactions are processed across the system's layers. For example, when a user performs a "Like" action, the event triggers the `SwipeService`, which validates and records the swipe, checks for mutual matches, and—if both users liked each other—creates a new `Connection` entity in the `Matches` table.

This flow contextualizes the domain-centric method signatures presented in the next section, showing how each function contributes to the overall behavior of the system.

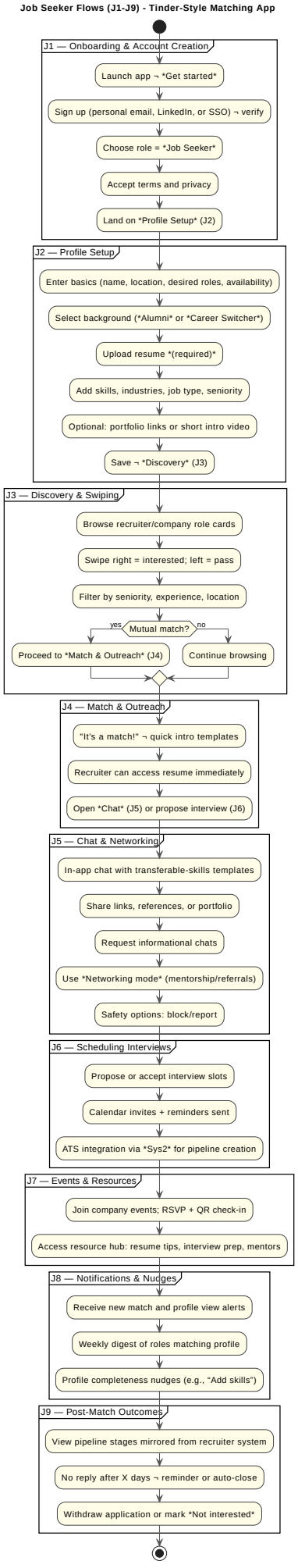

Job Seeker Flows (J1-J9) - Tinder-Style Matching App
J1 — Onboarding & Account Creation
Launch app ¬ *Get started*
Sign up (personal email, LinkedIn, or SSO) ¬ verify
Choose role = *Job Seeker*
Accept terms and privacy
Land on *Profile Setup* (J2)
J2 — Profile Setup
Enter basics (name, location, desired roles, availability)
Select background (*Alumni* or *Career Switcher*)
Upload resume *(required)*
Add skills, industries, job type, seniority
Optional: portfolio links or short intro video
Save ¬ *Discovery* (J3)
J3 — Discovery & Swiping
Browse recruiter/company role cards
Swipe right = interested; left = pass
Filter by seniority, experience, location
Mutual match?
yes
no
Proceed to *Match & Outreach* (J4)
Continue browsing
J4 — Match & Outreach
"It's a match!" ¬ quick intro templates
Recruiter can access resume immediately
Open *Chat* (J5) or propose interview (J6)
J5 — Chat & Networking
In-app chat with transferable-skills templates
Share links, references, or portfolio
Request informational chats
Use *Networking mode* (mentorship/referrals)
Safety options: block/report
J6 — Scheduling Interviews
Propose or accept interview slots
Calendar invites + reminders sent
ATS integration via *Sys2* for pipeline creation
J7 — Events & Resources
Join company events; RSVP + QR check-in
Access resource hub: resume tips, interview prep, mentors
J8 — Notifications & Nudges
Receive new match and profile view alerts
Weekly digest of roles matching profile
Profile completeness nudges (e.g., "Add skills")
J9 — Post-Match Outcomes
View pipeline stages mirrored from recruiter system
No reply after X days ¬ reminder or auto-close
Withdraw application or mark *Not interested*

Figure 8. Jobseeker flows diagram showing jobseeker interactions (search, view jobs, apply, save, follow-up) and how those actions map to backend services such as profile updates, application repository writes, and notification triggers.

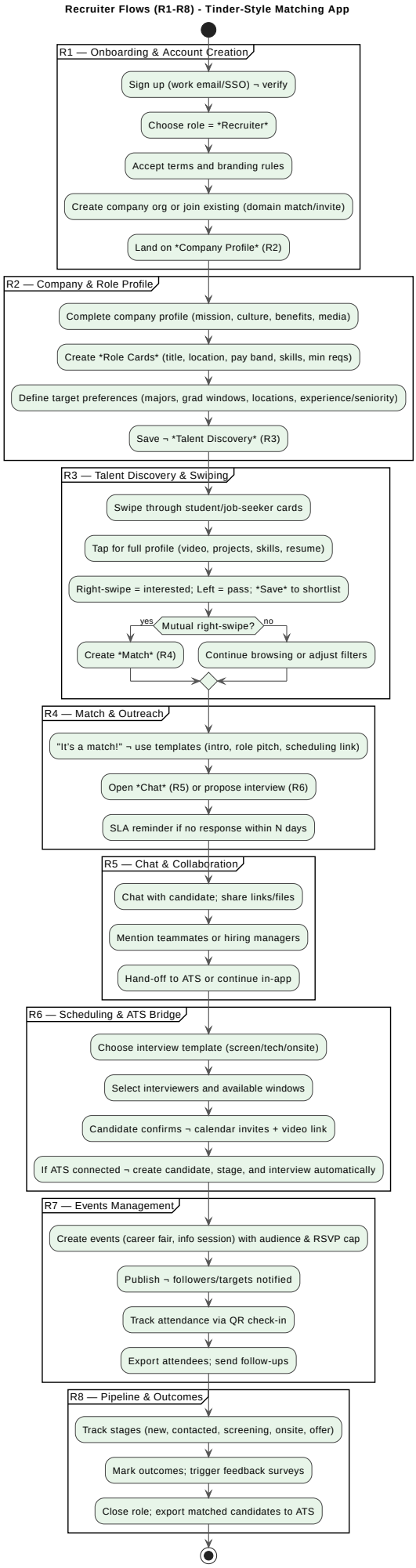

Recruiter Flows (R1-R8) - Tinder-Style Matching App
R1 — Onboarding & Account Creation
Sign up (work email/SSO) ¬ verify
Choose role = *Recruiter*
Accept terms and branding rules
Create company org or join existing (domain match/invite)
Land on *Company Profile* (R2)
R2 — Company & Role Profile
Complete company profile (mission, culture, benefits, media)
Create *Role Cards* (title, location, pay band, skills, min reqs)
Define target preferences (majors, grad windows, locations, experience/seniority)
Save ¬ *Talent Discovery* (R3)
R3 — Talent Discovery & Swiping
Swipe through student/job-seeker cards
Tap for full profile (video, projects, skills, resume)
Right-swipe = interested; Left = pass; *Save* to shortlist
Mutual right-swipe?
yes
no
Create *Match* (R4)
Continue browsing or adjust filters
R4 — Match & Outreach
"It's a match!" ¬ use templates (intro, role pitch, scheduling link)
Open *Chat* (R5) or propose interview (R6)
SLA reminder if no response within N days
R5 — Chat & Collaboration
Chat with candidate; share links/files
Mention teammates or hiring managers
Hand-off to ATS or continue in-app
R6 — Scheduling & ATS Bridge
Choose interview template (screen/tech/onsite)
Select interviewers and available windows
Candidate confirms ¬ calendar invites + video link
If ATS connected ¬ create candidate, stage, and interview automatically
R7 — Events Management
Create events (career fair, info session) with audience & RSVP cap
Publish ¬ followers/targets notified
Track attendance via QR check-in
Export attendees; send follow-ups
R8 — Pipeline & Outcomes
Track stages (new, contacted, screening, onsite, offer)
Mark outcomes; trigger feedback surveys
Close role; export matched candidates to ATS

Figure 9. Recruiter flows diagram illustrating recruiter workflows (post job, screen applicants, schedule interviews, take notes), ATS interactions, prescreen operations, and note persistence in repositories.

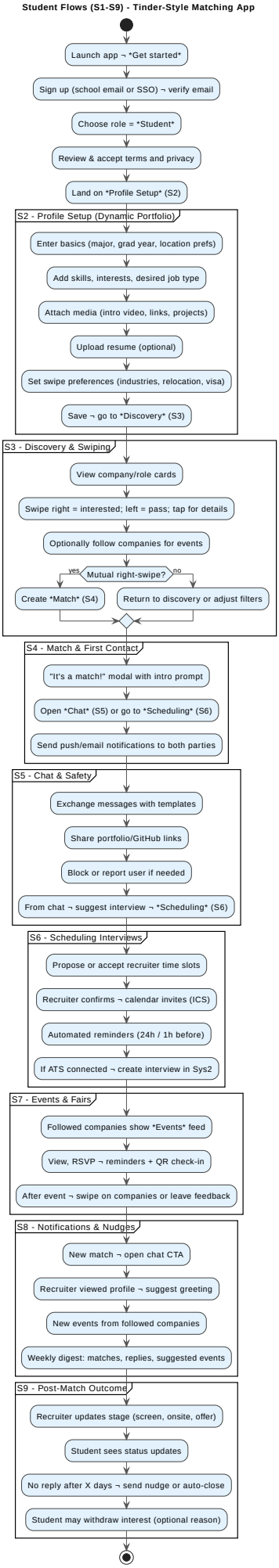

Student Flows (S1-S9) - Tinder-Style Matching App
Launch app ¬ *Get started*
Sign up (school email or SSO) ¬ verify email
Choose role = *Student*
Review & accept terms and privacy
Land on *Profile Setup* (S2)
S2 - Profile Setup (Dynamic Portfolio)
Enter basics (major, grad year, location prefs)
Add skills, interests, desired job type
Attach media (intro video, links, projects)
Upload resume (optional)
Set swipe preferences (industries, relocation, visa)
Save ¬ go to *Discovery* (S3)
S3 - Discovery & Swiping
View company/role cards
Swipe right = interested; left = pass; tap for details
Optionally follow companies for events
Mutual right-swipe?
yes
no
Create *Match* (S4)
Return to discovery or adjust filters
S4 - Match & First Contact
"It's a match!" modal with intro prompt
Open *Chat* (S5) or go to *Scheduling* (S6)
Send push/email notifications to both parties
S5 - Chat & Safety
Exchange messages with templates
Share portfolio/GitHub links
Block or report user if needed
From chat ¬ suggest interview ¬ *Scheduling* (S6)
S6 - Scheduling Interviews
Propose or accept recruiter time slots
Recruiter confirms ¬ calendar invites (ICS)
Automated reminders (24h / 1h before)
If ATS connected ¬ create interview in Sys2
S7 - Events & Fairs
Followed companies show *Events* feed
View, RSVP ¬ reminders + QR check-in
After event ¬ swipe on companies or leave feedback
S8 - Notifications & Nudges
New match ¬ open chat CTA
Recruiter viewed profile ¬ suggest greeting
New events from followed companies
Weekly digest: matches, replies, suggested events
S9 - Post-Match Outcome
Recruiter updates stage (screen, onsite, offer)
Student sees status updates
No reply after X days ¬ send nudge or auto-close
Student may withdraw interest (optional reason)

Figure 10. Student flows diagram capturing student-specific steps including event check-in, resume tailoring, booth pitch, informal networking, and multichannel application submission, mapped to system events and follow-up operations.

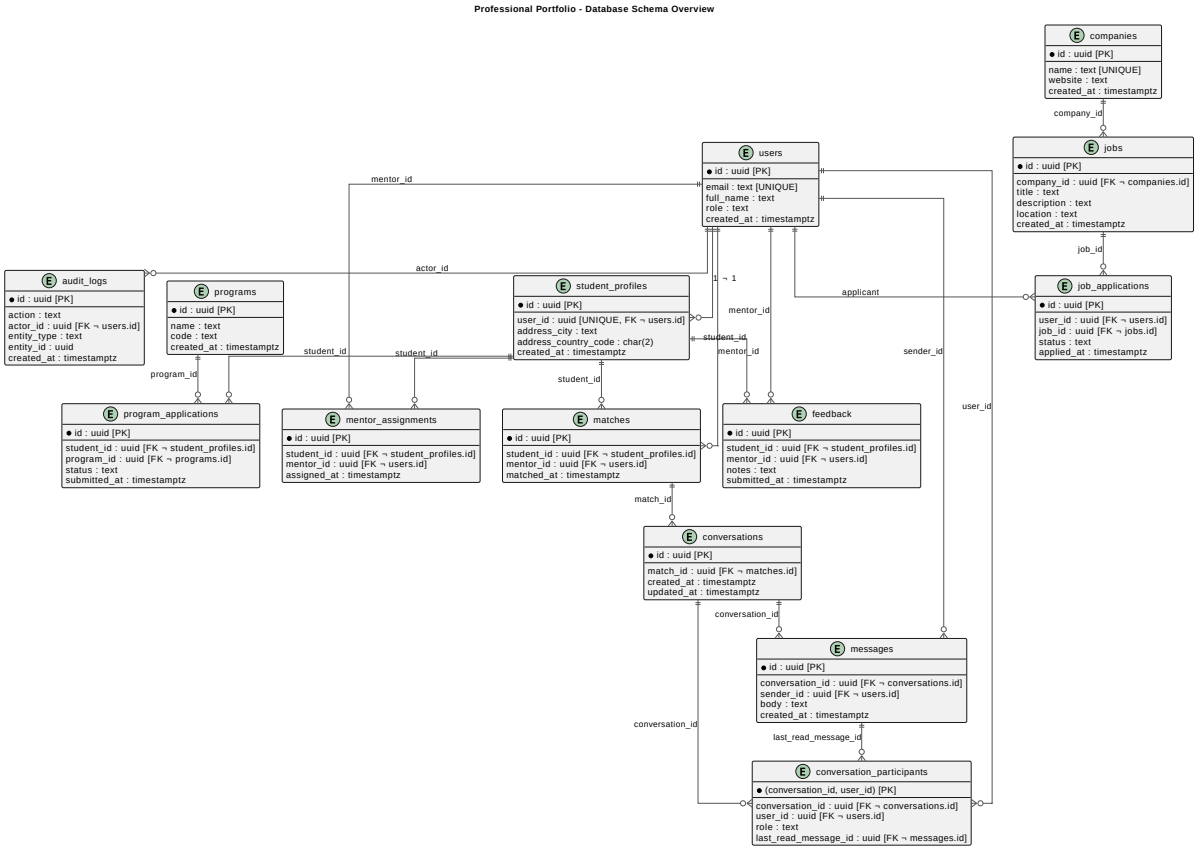


Figure 11. Database schema diagram (ER) showing core domain tables — users, student_profiles, programs, program_applications, mentor_assignments, matches, conversations, messages, conversation_participants, feedback, audit_logs, companies, jobs, and job_applications — with primary keys, foreign keys, relationships, and key indices used for deduplication, matching, and conversation threading.

2.3.2. Selected Fragments of the Implementation

The implementation fragments presented here illustrate how the proposed system realizes its core concepts—profiles, swiping, matching, messaging, and event participation—within the domain of student-recruiter interactions. They are not exhaustive; instead, they show how selected components are translated into concrete structures and operations, maintaining a clear separation between the core domain concepts and their presentation layer realizations.

~~The implementation fragments presented here illustrate how the proposed system realizes its core concept rofiles, swiping, matching, messaging, and event participatio ithin the domain of student-recruiter interactions. They are not exhaustive; instead, they show how selected components are translated into concrete structures and operations.~~

Domain-Centric Function Signatures

Domain-Centric Function SignaturesIn Flutter/Dart, we express domain contracts through method signatures and service interfaces. These serve as contracts between the UI layer, application logic,

and backend services. The functions below operate on the $\mathbf{Profile}$ entity, which represents the core data, not on the presentation $\mathbf{ProfileCard}$ widget.

~~In Flutter/Dart, we express domain contracts through method signatures and service interfaces. These serve as contracts between the UI layer, application logic, and backend services.~~

```
// Fetch the next profile card for a user□s deck
Future<ProfileCard?> getNextProfile(String userId);

// Record a swipe action, either Like or Pass
Future<Result<void>> processSwipe(String userId, String profileId, SwipeDirection
direction);

// Determine if a mutual Like exists
Future<Match?> checkForMatch(String userId, String profileId);

// Create a persistent connection once a Match occurs
Future<Connection> createConnection(Match match);

// Exchange a message between two Matched users
Future<Result<Message>> sendMessage(String connectionId, String userId, String
content);
```

Each function encodes a clear business rule. For example, `processSwipe` ensures that only valid swipe actions are stored, while `checkForMatch` guards the creation of Matches until both parties express interest.

User Interface Complement

A conceptual $\mathbf{ProfileCard}$ widget in Flutter may look like this. This shows how the domain concept ($\mathbf{Profile}$) is passed to a $\mathbf{presentation-layer}$ widget ($\mathbf{ProfileCard}$) for display and interaction. The widget itself belongs to the UI, but it depends on the domain's $\mathbf{Profile}$ data structure.

~~A conceptual **Profile Card** widget in Flutter may look like this. This shows how domain concepts (like `ProfileCard`) translate directly into Flutter widgets.:~~

```
class ProfileCard extends StatelessWidget {
  final Profile profile;
  final VoidCallback onLike;
  final VoidCallback onPass;
  final VoidCallback onMoreInfo;

  const ProfileCard({
    required this.profile,
    required this.onLike,
    required this.onPass,
    required this.onMoreInfo,
    super.key,
```

```
  });

  @override
  Widget build(BuildContext context) {
    return Card(
      margin: const EdgeInsets.all(12),
      child: Column(
        children: [
          Text(profile.name, style: Theme.of(context).textTheme.headline6),
          Text(profile.details),
          Row(
            mainAxisAlignment: MainAxisAlignment.spaceEvenly,
            children: [
              IconButton(icon: Icon(Icons.close), onPressed: onPass),
              IconButton(icon: Icon(Icons.favorite), onPressed: onLike),
              IconButton(icon: Icon(Icons.info), onPressed: onMoreInfo),
            ],
          )
        ],
      ),
    );
  }
}
```

Chapter 3. Analytic Part

3.1. Concept Analysis

This section refines the raw material from the "Domain Rough Sketch" into a set of domain concepts, abstractions, and relationships, following the guidelines for concept analysis. The process began by reviewing unprocessed notes and quotes, then abstracting recurring ideas into general concepts, clarifying their meaning, and documenting how they relate. This analysis provides a foundation for requirements and system design.

3.1.1. Method

We systematically reviewed the rough sketch, highlighting recurring themes, terms, and pain points. For each, we considered whether it represented a domain concept, event, or behavior, and how it might generalize beyond the specific example. Where terms were ambiguous or used differently by stakeholders, we clarified or resolved them for consistency.

3.1.2. Key Concepts Identified

Queue/Waiting Time: Multiple students mentioned long waits at job fairs. We abstract this as the concept of a "queue," representing bottlenecks in event based interactions.

Profile: resumes, portfolios, and company descriptions are generalized as "profiles", structured representations of actors in the domain. We distinguish between candidate and recruiter profiles.

Match and Connection: Mutual interest is abstracted as a "Match." If acted upon, this becomes a "connection", a persistent relationship enabling further interaction.

Discovery Feed: Reviewing multiple candidates or companies is generalized as a "discovery feed," the set of profiles available for evaluation.

Event: Job fairs, meetups, and interviews are all instances of "events," structuring interactions in time and space.

Note Taking: Recruiter’s use of spreadsheets and notes is abstracted as "record keeping," essential for memory and making decisions in high volume interactions.

Application: Submitting interest through various channels is generalized as an "application," linking candidates to opportunities.

3.1.3. Clarifications and Resolutions

- The term "candidate" is used broadly to mean any job seeker, regardless of experience.
- "Recruiter" refers specifically to the human actor representing an employer, not the company itself.
- "Profile card" is distinguished from "profile" as the interactive, condensed representation used in the discovery process.

- "Match" is an event, while "connection" is a state that persists after a Match.

3.1.4. Relationships and Abstractions

- Candidates build and share their profiles, skills, and qualifications with recruiters.
- Recruiters evaluate candidates based on these profiles, often within the context of an event.
- When mutual interest is signaled, a Match is formed, which can become a connection.
- Connections enable further actions, such as messaging or scheduling interviews.
- Events provide the environment where many of these interactions are initiated.
- Record keeping and note taking support making decisions and memory throughout the process.

3.1.5. Reasoning and Process

This analysis was grounded in the raw observations and quotes from the rough sketch, ensuring that abstractions are traceable to real domain phenomena. Ambiguities were resolved by referencing both stakeholder language and the needs of the requirements phase. By documenting this process, we ensure that the resulting vocabulary is both consistent and shared across the team.

Through this analysis, the scattered ideas from the descriptive phase are distilled into a structured vocabulary. These concepts and relationships now form a shared foundation for the requirements and system design phases that follow.

3.2. Validation and Verification

The purpose of this section is to document how the team will validate and verify the domain concepts, requirements, and design decisions.

- **Validation**: ensuring that what we documented reflects the real-world domain (student–recruiter interactions, job fairs, portfolio showcases, and online hiring practices).
- **Verification**: ensuring that the documentation is internally consistent, complete, and aligned with the project's goals.

3.2.1. Validation Strategy

Validation activities focus on ensuring that our assumptions, workflows, and requirements accurately reflect real-world recruiting, mentoring, and hiring practices. Rather than relying solely on abstract flows, this section grounds validation through concrete, scenario-based walkthroughs involving realistic personas and contexts.

- **Literature and Online References** Foundational validation draws from established data and professional frameworks:
  - The **National Association of Colleges and Employers (NACE)** reports that **over 80% of employers evaluate student portfolios or online profiles as part of their screening process**.
  - **LinkedIn's Global Talent Trends** emphasize that **skills, projects, and practical**

experiences now outweigh traditional credentials.

    - **University career centers** (e.g., UC Berkeley, Purdue, UPRM) encourage students to include **artifacts, capstone projects, and self-assessments** in their portfolios, aligning with our design decisions for portfolio sections and recruiter search filters.

- **Scenario Walkthroughs (Concrete Personas for Validation)** Early walkthroughs were initially too abstract, which limited the kind of stakeholder insights we could extract. The updated validation strategy introduces **detailed, narrative-driven personas** that simulate realistic interactions between users and the system. These narratives promote richer discussions with stakeholders by presenting grounded, relatable contexts.

Scenario 1: Abdul (Student) and Corey (Recruiter) — Capstone Connection

Abdul is a 5th-year Mechanical Engineering student finishing his final semester. He is taking **Thermodynamics**, **Pottery II**, and the **Capstone Design** course. His capstone project, **"Predicting Beer Pleasantness from Brewing Parameters,"** leverages data modeling and process optimization — experience relevant to industrial process control.

Corey is a recruiter for **Lilly**, seeking candidates for a process control engineering position. Using the Professional Portfolio platform, Corey filters candidates by "data analysis" and "process modeling." Abdul's profile appears, but his capstone project is missing. Abdul updates his profile, adding a brief description and project artifacts. Once saved, the system automatically updates search indices. Corey's filter now surfaces his profile again, and she swipes right to indicate interest.

Validation Points: - Tests whether **profile updates** propagate correctly to recruiter searches. - Validates **event-driven refresh triggers** in recruiter dashboards. - Checks whether **project tagging and search weighting** align with recruiter expectations. - Reveals potential enhancement: recruiter alerts for updated profiles that match their saved filters.

Scenario 2: Aisha (Mentor) and Daniel (Student) — Mentorship Match Validation

Aisha is a volunteer mentor from the tech industry who joined the platform through a university partnership. She specifies her domain as "data science and product analytics." Daniel, a computer engineering student, signs up for mentoring. He lists his interests as "AI systems," "data visualization," and "career readiness."

The system's mentor matching service pairs them based on overlapping interests. During the first meeting, Daniel mentions wanting to add his internship project portfolio. Aisha advises him to include visualizations of his code metrics. Later, Daniel updates his portfolio accordingly.

Validation Points: - Confirms **mentor matching criteria** reflect relevant skill intersections. - Tests **feedback loops** where mentors influence students' portfolio enrichment. - Surfaces design opportunities for **mentor notes or endorsement badges** to appear on student profiles.

Scenario 3: Rafael (Recruiter) — Shortlisting and Feedback Integration

Rafael is a recruiter from a mid-sized design firm reviewing candidates for UX positions. He uses the filtering tool to shortlist candidates by "portfolio completeness > 80%" and "projects with client collaboration." He finds Emma's profile and likes her work but notices missing context for one

project. He leaves structured feedback through the recruiter interface, suggesting more detail on design impact.

A few days later, Emma edits her project entry, expanding her case study. The system notifies Rafael that an update was made. This re-engagement loop allows him to reopen her profile and mark it as "ready for review."

Validation Points: - Tests **feedback submission and notification logic** between recruiters and students. - Validates **recruiter engagement retention** through automatic follow-ups. - Identifies possible feature expansion for **feedback analytics dashboards**.

Scenario 4: Lucia (Career Advisor) — Oversight and Data Consistency

Lucia, a career center advisor, monitors how active recruiters interact with student portfolios. She notices inconsistent data entry (some portfolios lack tags). She requests that the platform provide a "data completeness report" per department. The request leads to the idea of integrating a **Portfolio Health Metric** — a completeness score visible to both students and advisors.

Validation Points: - Ensures **role-based access control** allows advisors to view, not alter, student data. - Encourages new features that support **data quality monitoring** for institutional users.

- **Stakeholder Proxies and Next Steps** Until direct recruiter and mentor interviews begin, **career center advisors** and **industry-aligned professors** serve as proxies for validation. These personas, rooted in observed academic and recruitment contexts, will evolve as we collect field data from pilot users. Future steps include:
  - Conducting **cognitive walkthroughs** with recruiters and students to validate UI intuitiveness.
  - Running **A/B tests** on search filters and project visibility ranking.
  - Integrating qualitative feedback into the next iteration of **matching and notification workflows**.

3.2.2. Verification Strategy

Verification ensures that the documentation is internally consistent and traceable across milestones.

- **Peer Reviews (Planned)** Each section of the documentation (terminology, requirements, narrative) will be reviewed by a different team member. Early reviews already revealed ambiguities between the terms "candidate" and "student," which were clarified.
- **Checklists** A milestone-based checklist will be used:
  - Every requirement uses only defined terms.
  - Requirements map to at least one domain concept.
  - No contradictions between terminology and requirements.
  - Requirements are measurable and testable (planned for Milestone 2).
- **Traceability Matrix (Planned)** A lightweight traceability matrix will link:

    - **Requirements → Goals** (does each requirement support at least one documented goal?).
    - **Requirements → Terminology** (is consistent terminology used?).
- **Walkthroughs of Documentation** We will simulate “reader walkthroughs,” where one team member acts as an external reviewer and checks whether requirements can be traced back to definitions. This approach helps identify undefined or ambiguous terms.

Chapter 4. Specific Class Topics

4.1. Communication and the Use of Language

4.1.1. Knowledge Crunching for Recruiter Needs

NOTE	Objective: Understand what recruiters actually need from messaging and notifications to guide near-term design/implementation.

One-Page Interview Guide

- Which events are immediate vs batch later? Give concrete examples.
- During quiet hours, what must still break through (by role/stage)?
- Preferred channel per event (push, in-app, email, SMS opt-in) and why.
- Timing windows: reminders (T-24h/T-2h), follow-ups after 48–72h silence, offer deadlines.
- Inbox filters you must have (Interview Today, Needs Feedback, New Priority Match, At-Risk/Idle X days).
- Triage: labels, urgency flags, snooze/mute rules.
- Exact deep-link target on open (thread top, interview card, offer card, note anchor).
- Receipts/“seen” policy (opt-in? per thread/org?). Etiquette concerns.
- Internal mentions: who gets notified and when (@roles, offer steps).
- Automatic follow-ups (first/second nudge cadence, templates).
- Compliance/consent: time zones, opt-in for SMS/email, do-not-disturb, retention.
- One thing current tools get wrong about notifications—and how to fix it.

Interviews

Interview 01: Mid-market Tech Recruiter · Date: 2025-10-05 · Mode: phone (28 min) · Region: urban + suburban sites · Anonymized

Answer 1: Immediate vs batch. *Immediate:* same-day interviews, <24h reschedules, offer steps. *Batch:* new applicants, minor status changes.

Answer 2: Quiet hours. 7pm–7am local; exceptions: <24h reschedule, offer accept/expire.

Answer 3: Channels. Urgent → push + in-app (SMS if candidate opted-in); routine → in-app + email digest.

Answer 4: Timing. Reminders T-24h/T-2h; feedback prompt 2h post-interview; no-reply nudges at 48h/96h.

Answer 5: Filters. Interview Today, Needs Feedback, New Priority Match, Idle 3+ days.

Answer 6: Triage. Urgent flag; snooze until 08:00; mute threads after closure.

Answer 7: links. Open the interview card pinned inside the thread.

Answer 8: Receipts. Delivery timestamp OK; “seen” off by default (adds pressure).

Answer 9: Internal. Hiring Manager on interview feedback; Finance only on offer.

Answer 10: Automation. “Thanks—feedback soon” to candidates; 48/96h nudges to keep momentum.

Answer 11: Compliance. Respect candidate time zone; store per-channel consent; honor DND.

Answer 12: tools. Too many pings; batch low-signal changes and keep deep links precise.

Structured Notes (Interview 1):

Theme	Signals / Details
Urgent	Interview day, reschedule<24h, offer
Batch	New applicant, non-critical status change
Policies	Per-thread quiet hours + role overrides
Navigation	Deep link → interview/offer card
Etiquette	Delivery receipts over “seen”
Automation	48/96h nudges; 2h feedback prompt

Interview 2: Agency Recruiter (multi-timezone) · Date: 2025-10-06 · Mode: video (28 min) · Region: urban + suburban sites · Anonymized

Answer 1: Immediate vs batch. Immediate: offer steps, day-of schedule changes. Batch: sourcing/status updates.

Answer 2: Quiet hours. After 7pm local; override only for <24h reschedules/offer deadlines.

Answer 3: Channels. Urgent → push; day-of SMS if opted-in; daily email digest for summaries.

Answer 4: Timing. T-24h/T-2h reminders; auto re-invite at 72h no-reply.

Answer 5: Filters. Interview Today, Offer Pending, At-Risk (no response 96h).

Answer 6: Triage. Labels: Urgent / Follow-up / Waiting; snooze per thread.

Answer 7: Deep links. Interview details with join link and call button.

Answer 8: Receipts. Opt-in per thread; org default is off.

Answer 9: Internal. @Legal/@Finance only during offer; no alerts for routine notes.

Answer 10: Automation. Post-interview “thank you” + 48/96h nudges.

Answer 11: Compliance. Regional DND and recorded consent (SMS/email).

Answer 12: Gap in tools. Alerts are not role-aware; everything notifies everyone.

Structured Notes (Interview 2):

Theme	Signals / Details
Urgent	Reschedule<24h, offer, interview reminder
Batch	Sourcing and stage changes
Overrides	Offer steps ignore quiet hours for recruiter/manager
Deep link	Interview details with join link
Automation	72h re-invite; At-Risk at 96h
Consent	Per-channel opt-ins saved

Synthesis

What recruiters actually need from chat/notifications?

- Urgency rules separate true interrupts (reschedule<24h, offer, day-of interview) from batchable noise (status/sourcing).
- Per-thread quiet hours with role-based overrides for offer & day-of scheduling.
- Channels by event (push/in-app urgent; email digest for low-signal; SMS only with opt-in).
- Reliable deep links to the exact action (Interview card, Offer card, Note anchor).
- Inbox controls (Interview Today, Needs Feedback, New Priority Match, At-Risk/Idle).
- Polite automation (48/96h nudges; feedback prompt 2h; 72h re-invite).
- Etiquette & consent defaults (delivery receipts vs “seen”; stored per-channel consent).
- Outcomes: fewer no-shows, faster time-to-respond, clear audit of what alerted whom and why.

Event Taxonomy & Matrix

Event	Actor	When	Priority	Channel	Deep Link
Interview Reminder	Candidate, Recruiter	T-24h, T-2h	High	Push + In-app	Interview card
Reschedule/Cancel <24h	Both	Instant	Break Quiet Hours	Push + SMS (opt-in)	Interview card (new time)
Offer Extended	Candidate, Recruiter	Instant	High	Push + Email	Offer card
Offer Deadline T-48h	Candidate, Recruiter	T-48h	High	Push + In-app	Offer card (deadline)

Event	Actor	When	Priority	Channel	Deep Link
New Priority Match	Recruiter	Instant	High	Push + In-app	Profile → thread
No Reply 48h/96h	Recruiter	Daily batch	Medium	In-app + badge	Thread with “Nudge”
Internal @Mention	Mentioned role	Instant	Normal	In-app + Email	Note anchor
Doc Request	Candidate	Daily batch	Low	Email	Files tab
Snooze Ends	Recruiter	At wake time	Low	In-app	Thread top

Prioritization

Must

- Policy engine for urgency & quiet hours (thread/role aware)
- Reliable deep links (interview/offer/note anchors)
- RSVP + calendar integration
- Morning digest (07:30 local) for low-signal updates
- Auto-nudge at 48/96h silence
- Internal @mentions with anchors

Should

- Opt-in delivery receipts and SLA badges
- Candidate quiet hours and consent per channel
- Role-based overrides for urgent events

Could

- Stage-aware bundling of notifications
- Intent/sentiment hints in threads

Won’t (now)

- Cross-org notification analytics

Implementable Requirements

R1 — Policy Engine for Urgency & Quiet Hours

```
Feature: Notification policy engine

Scenario: Reschedule within 24 hours breaks quiet hours
  Given the thread has quiet hours from 19:00 to 07:00
  And an interview is rescheduled 12 hours before start
  When the event is processed at 22:00
  Then a HIGH priority push notification is sent
  And the deep link opens the interview card

Scenario: Non-urgent updates during quiet hours are batched
  Given quiet hours are active
  When a status change event occurs
  Then no push is sent
  And the update appears in the 07:30 digest
```

R2 — Interview RSVP + Calendar Deep Link

```
Feature: RSVP and calendar

Scenario: Candidate receives invite
  Given an interview invite is sent
  When the candidate opens the notification
  Then the app shows RSVP buttons
  And an .ics file is available
  And the meeting link is visible
```

R3 — Daily Digest at 07:30 Local

```
Feature: Morning digest

Scenario: Recruiter starts day
  Given there are interviews today and 48h+ unreplied threads
  When it is 07:30 local time
  Then a digest groups interviews, unreplied threads, and new priority matches
  And each item deep links to the exact context
```

R4 — Auto-Nudge After Silence (48h/96h)

```
Feature: Auto follow-up

Scenario: First nudge at 48h
  Given a candidate has not replied for 48 hours
  When automation runs
  Then a polite nudge is posted in the thread
```

```
Scenario: At-risk at 96h
  Given no reply after the first nudge
  When 96 hours elapse
  Then a second nudge is posted
  And the thread is labeled "At Risk"
```

R5 — Thread-Level Mute/Snooze

```
Feature: Snooze thread

Scenario: Snooze until tomorrow 08:00
  Given a recruiter snoozes a thread until 08:00 tomorrow
  Then no notifications are sent for that thread until 08:00
  And a "Snoozed" badge is visible
```

R6 — Internal @Mentions with Anchors

```
Feature: Internal mentions

Scenario: Mention hiring manager
  Given a note mentions the hiring manager
  Then the manager receives a notification
  And opening it jumps to the note anchor in the thread
```

R7 — Delivery Receipts + SLA Badges (opt-in)

```
Feature: Delivery receipts and SLA

Scenario: SLA risk after threshold
  Given the organization SLA is response within 24 hours
  And a message has been delivered for 24 hours without reply
  Then the thread shows "SLA at risk"
```

Traceability — Pain → Event → Requirement:

Pain Point	Event(s)	Requirement(s)
Too many alerts / noise overload	Reschedule<24h, Interview reminders, Offer updates	R1 (Policy Engine), R3 (Morning Digest)
Missed follow-ups / dropped threads	No Reply 48h/96h	R4 (Auto-Nudge After Silence)

Pain Point	Event(s)	Requirement(s)
Wrong screen after tapping notification	All (Interview, Offer, Mention)	R2 (Deep Links + RSVP/Calendar)
Coordination gaps between roles	Internal @Mention	R6 (Internal @Mentions with Anchors)
Pressure from read receipts	Any (messages, updates)	R7 (Delivery Receipts opt-in)
Need for silent focus time	All (Batch, low-priority)	R1 (Policy Engine + Quiet Hours)
Inconsistent candidate updates	Interview reminder, Offer deadline	R2 (RSVP/Calendar), R3 (Digest)

References

LinkedIn. (n.d.). Alerts and notifications. Recruiter Help. https://www.linkedin.com/help/recruiter/topic/a100022

LinkedIn. (n.d.). InMail and Inbox. Recruiter Help. https://www.linkedin.com/help/recruiter/topic/a52

Workable. (n.d.). Scheduling events overview. Workable Help Center. https://help.workable.com/hc/en-us/articles/115013135088-Scheduling-events-overview

Workable. (n.d.). Do candidates receive reminder emails for scheduled events? Workable Help Center. https://help.workable.com/hc/en-us/articles/32871072712855-Do-candidates-receive-reminder-emails-for-scheduled-events

Indeed for Employers. (2025, May 27). How to message candidates on Indeed: A guide for employers. https://www.indeed.com/hire/resources/howtohub/indeed-messaging

Indeed for Employers. (2024, November 4). 6 texting and chatting rules with candidates. https://www.indeed.com/hire/c/info/texting-chatting-rules-with-candidates

04CommunicationAndTheUseOfLanguage.pdf — Ubiquitous Language; aligning terms across interviews, requirements, and data/model.

4.1.2. Minimal vs Enriched UML

In the enriched model the operation of accepting or swiping right to allow communication between recruiter and job seeker from the perspective of the recruiter, with the assumption that job seekers have accepted or “swiped right” a job or program that this recruiter manages/is a mentor of. It is decomposed into the layers that are used in the order they are used by the system. It shows 2 user scenarios, one were the connection is available and access to the database is possible in the moment it is requested or “happy path” and another when the access is not possible at that moment which makes the system look for a backup or cache of the matches that could have been retrieved in a previous fetch and if that also fails it will store the user interaction to update database with the accept flag for the role of the current user on the corresponding user of opposite role.

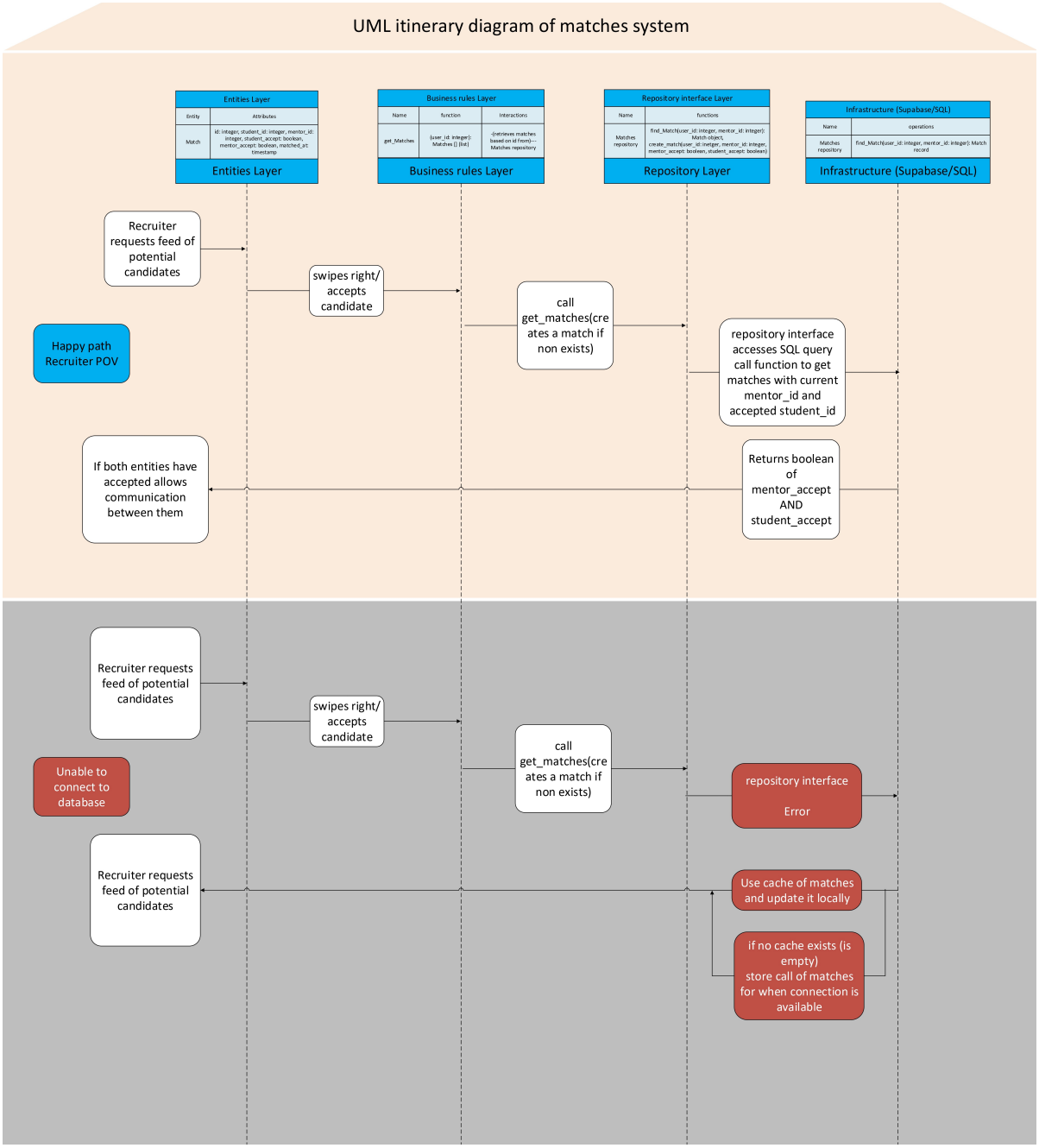


Figure 12. Itinerary of recruiter scenario showing the various stages and interactions involved in the recruitment process.

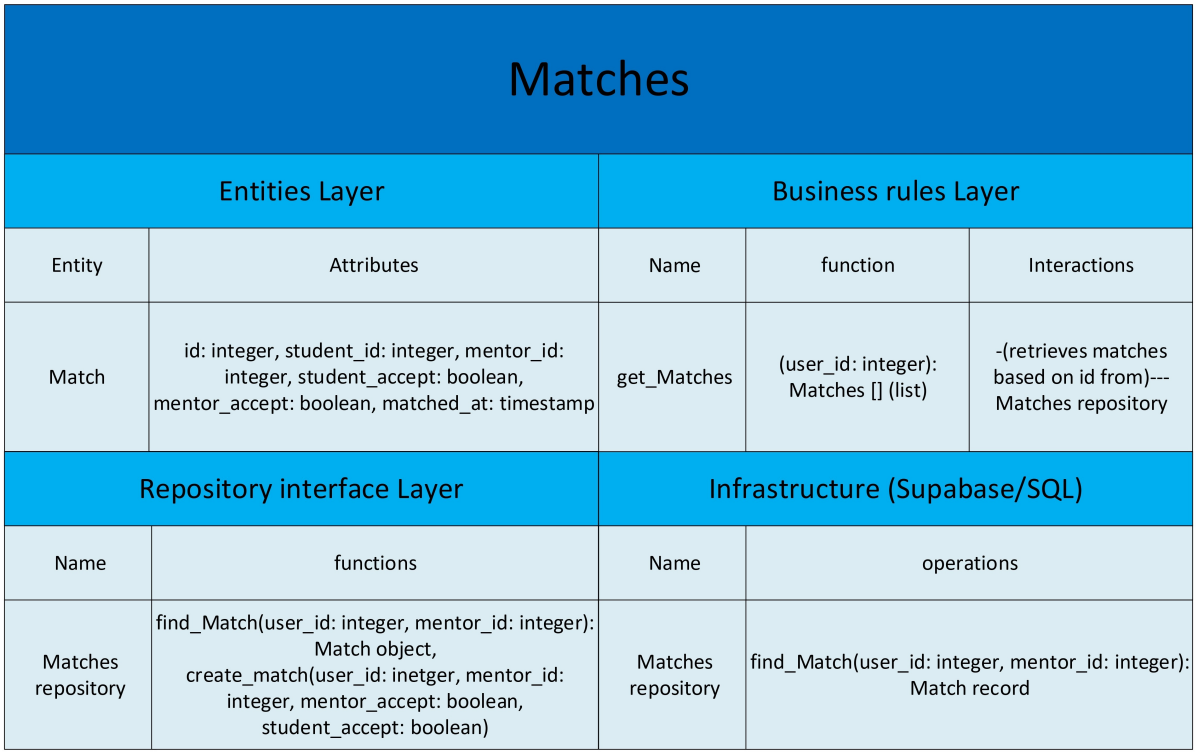


Figure 13. UML diagram showing the entities involved in the matching process between recruiters and students.

4.2. Isolating the Domain & Expressing Models in Software

4.2.1. Domain Rules in Domain Layer

NOTE	Objective: Ensure all business rules are well established and available for developers to refer to and for potential clients to review.

Matching Policy

For a Match to occur between a Recruiter and a Student, there must first be a similarity between the Student's skillset and a Recruiter's JobOpening. If these skills are close enough, each user will have more chances to see each other in the App, and they can each accept or decline each other. If both users accept each other, a Match occurs, and they may begin interacting with each other via messages.

User Interactions

If a Match occurs, a Recruiter and a Student will begin direct contact through the App's messaging feature. Afterwards, they may arrange meetings and interviews by utilizing external tools to our App, such as Microsoft Teams, Zoom, In Person meetings, etc.

4.2.2. Modeling the Address as a Value Object

NOTE	Objective: Apply Section 4.3 (Isolating the Domain & Expressing Models in Software)

> by refactoring the `StudentProfile` aggregate to include `Address` as an immutable Value Object. This isolates the domain model, enforces invariants at construction, and maintains persistence independence through a repository layer.

Topic Overview

Section 4.3 emphasizes modeling the domain in code by identifying **entities**, **value objects**, and **aggregates** that mirror real-world concepts. Entities are defined by identity, while value objects are defined solely by their attributes and must remain immutable. The lecture also introduced factories and repositories as tools to keep domain logic clean and separated from technical infrastructure.

Application in the Project

In the Professional Portfolio System, the `StudentProfile` aggregate initially scattered raw address fields (`street`, `city`, `postal_code`, etc.), which led to duplicated logic and weak cohesion. Through knowledge-crunching and discussion, this revealed an **implicit concept** — the Address itself. The team made it explicit as a dedicated `Address` Value Object embedded within the `StudentProfile` aggregate root.

The `StudentProfileRepository` mediates between the domain and Supabase, exposing methods in terms of domain concepts (`getById`, `updateAddress`) rather than SQL statements. This structure follows the Dependency Inversion Principle, allowing the domain to remain pure while infrastructure adapts around it.

Implementation Summary

- **Aggregate Definition:** `StudentProfile` acts as the aggregate root, controlling all access and updates to its internal `Address` object.
- **Factory Construction:** `Address` uses a factory constructor that validates input and enforces domain rules before instance creation.
- **Value Object Semantics:** Two `Address` instances with the same attribute values are equal; all fields are `final`, ensuring immutability.
- **Repository Abstraction:** `StudentProfileRepository` translates between database rows and domain models (`_rowToAddress`, `_addressParams`) to maintain separation of concerns.
- **Persistence Shape:** Six flat columns were added to `student_profiles` for address data: `address_line1`, `address_line2`, `address_city`, `address_region`, `address_postal_code`, and `address_country_code`. A constraint ensures the `country_code` is always a two-letter uppercase ISO-3166 value.

Invariants and Guards

The following invariants must hold within the `Address` Value Object: * `line1`, `city`, and `postalCode` cannot be empty. * `countryCode` must match `/^[A-Z]{2}$/`. * All string values are normalized (trimmed, single spaces). * Once created, an `Address` cannot be mutated; changes require a new instance.

Validation occurs at the boundary—user input is verified before `Address` construction to ensure all invariants hold. Profiles may temporarily lack an address (e.g., new user); null updates are

intentionally supported to clear address data safely.

Testing & Verification

A live Supabase integration test demonstrated: . The user authenticated and updated their own `StudentProfile` with a valid `Address`. . The repository persisted and retrieved the same value object correctly. . Null updates safely cleared all address columns. . Row-Level Security (RLS) correctly restricted updates to the profile owner.

All checks passed, confirming domain integrity, immutability, and persistence consistency.

Outcome and Reflection

Refactoring `Address` as a Value Object achieved: * **Isolation of the Domain:** Address logic resides solely in the model; database details are hidden behind the repository. * **Cohesion and Clarity:** Related primitives unified under one explicit concept. * **Data Integrity:** Invariants enforce correctness at creation time. * **Expressive Model:** Code now mirrors the domain language ("profile has an address") rather than database fields.

This task fully embodies the lecture's intent — expressing the domain through explicit modeling, factories, aggregates, and repositories — showing how theoretical design principles produce a practical, robust implementation.

Traceability to Lecture Concepts

Lecture Principle	Project Realization
Aggregate	`StudentProfile` is the root controlling `Address` updates.
Factory	`Address.factory()` enforces invariants before creation.
Repository	`StudentProfileRepository` isolates domain from Supabase.
Invariants	Validation of non-empty fields and ISO country code.
Making Implicit Concepts Explicit	Recognized Address as a hidden concept, now modeled explicitly.

References

- Schütz-Schmuck, Marko. **Isolating the Domain & Expressing Models in Software** (Lecture Notes, Section 4.3).

4.2.3. Factory Design for Match Creation

NOTE	Objective: Implement and document a `MatchFactory` that creates valid `Match` domain objects, enforcing invariants and preventing duplicates while providing a clear, testable construction boundary for the rest of the system.

Rationale: why a factory for Matches?

- Centralized validation: all rules that decide whether a Match can be created (valid identities,

eligibility, no conflicting matches) live in one place.

- Encapsulation of creation rules: higher-level components don't need to know the details of ID generation, timestamping, or duplicate checks.
- Maintainability: new rules (e.g., eligibility windows, verification status) are applied in the factory without changing callers.
- Traceability: the factory can attach creation metadata (actor, timestamp, rationale) to the created object for audit and debugging.

Design contract (short)

- Inputs: a `Student` instance and a `Recruiter` instance (domain entities), optional metadata (source, context).
- Outputs: `Result<Match>` — success with a `Match` object or failure with a typed error (ValidationError, DuplicateMatchError, AuthorizationError).
- Error modes: invalid/malformed entities, duplicates, domain guards (e.g., recruiter not active), persistence failures delegated to repositories.

MatchFactory responsibilities

1. Validate both participants are valid domain entities (non-null, correct type, active state).
2. Ensure no existing confirmed or pending Match conflicts (duplicate prevention).
3. Generate a unique Match identifier (UUID) and canonical timestamps.
4. Attach provenance metadata (createdBy, createdAt, source) to the Match.
5. Return a clear Result type to the caller with success or an explanation of failure.

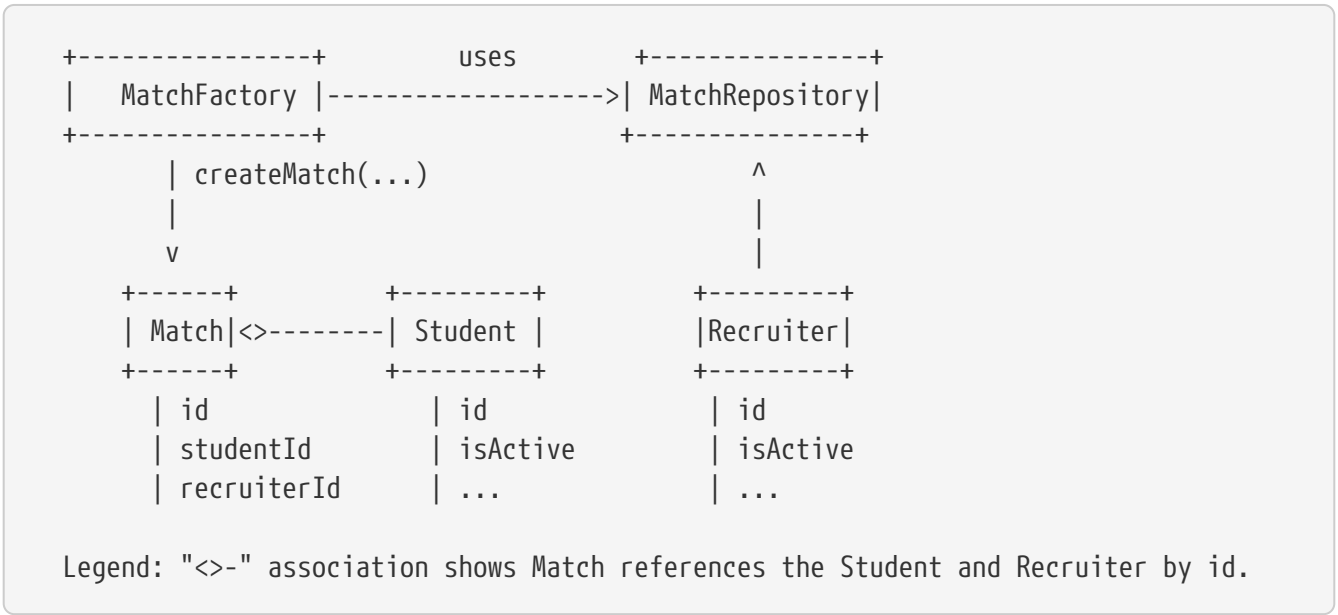
Example Dart fragment

```
class MatchFactory {
  final MatchRepository _repo;
  MatchFactory(this._repo);

  Future<Result<Match>> createMatch({
    required Student student,
    required Recruiter recruiter,
    String? source,
  }) async {
    // 1. Basic validation
    if (!student.isActive || !recruiter.isActive) {
      return Result.failure(ValidationError('Both parties must be active'));
    }

    // 2. Duplicate prevention
    final exists = await _repo.existsBetween(student.id, recruiter.id);
    if (exists) return Result.failure(DuplicateMatchError());

    // 3. ID generation and creation
```

```
    final match = Match(
      id: Uuid().v4(),
      studentId: student.id,
      recruiterId: recruiter.id,
      createdAt: DateTime.now().toUtc(),
      status: MatchStatus.confirmed,
      provenance: MatchProvenance(source: source ?? 'swipe', createdBy: student.id),
    );

    // 4. Persist and return
    await _repo.save(match);
    return Result.success(match);
  }
}
```

Notes: The fragment uses a `Result<T>` wrapper for explicit success/failure handling and a `MatchRepository` to check duplicates and persist matches; this keeps persistence concerns out of the factory logic while allowing the factory to consult repositories as needed.

UML / Class diagram (ASCII)



Enforcing edge cases

- Null or incomplete entities: factory refuses and returns ValidationError.
- Inactive accounts: factory refuses to create matches for suspended or unverified participants.
- Duplicate attempts: repository-level `existsBetween` prevents duplicate creation; the factory returns DuplicateMatchError.
- Race conditions: callers should either rely on a transactional repository that enforces uniqueness at the DB level.

Traceability and logging

The factory should optionally emit a log event every time it creates or rejects a Match. The log includes the requested participants, the outcome, the reason (if rejected), and a timestamp. This supports audit requirements described in the Domain Requirements section.

Integration with Section 2.3.1 guidelines

The `MatchFactory` follows the Selected Fragments of Implementation approach by providing a clear domain-level API (factory method), keeping persistence behind a repository, and presenting a small set of method signatures that higher layers use. It avoids leaking technical details like SQL or UUID algorithms to callers.

Diagrammatic example: sequence (createMatch)

```
Client -> MatchFactory: createMatch(student, recruiter)
MatchFactory -> MatchRepository: existsBetween(studentId, recruiterId)?
alt exists == false
  MatchFactory -> Match: new Match(id=UUID(), ...)
  MatchFactory -> MatchRepository: save(match)
  MatchFactory -> Client: Result.success(match)
else
  MatchFactory -> Client: Result.failure(DuplicateMatchError)
end
```

4.2.4. SOLID-Oriented Software Design

At the **software architecture** level, the system is divided into three main components:

1. **Mobile Frontend (Flutter)** - Handles profile cards, swiping, and messaging.
2. **Application Backend** - Contains the business logic for swiping, matching, and event handling.
3. **Data Layer** - Manages persistence of profiles, swipes, Matches, and messages.

At the **software design** level, these components are realized through service classes and repositories. To ensure scalability and maintainability, the design applies the **SOLID principles**:

Single Responsibility Principle (SRP)

Each service is designed with a **single responsibility**, defined by **the reason it may need to change**. This principle ensures isolation between business rules and user interaction mechanisms.

Service	Domain or Application	Reason for Change (Responsibility)
`MatchingService`	Domain	**event:** `MatchingRuleUpdated` **guard:** Only changes when business criteria or scoring logic evolve. **outcome:** Adjusts how candidates and employers are matched based on updated domain rules.

Service	Domain or Application	Reason for Change (Responsibility)
`RecruiterService`	Domain	**event:** `RecruiterWorkflowModified` **guard:** Changes when recruitment workflows, job posting policies, or recruiter permissions evolve. **outcome:** Updates internal procedures for managing candidate pools or posting jobs.
`EmployerService`	Domain	**event:** `HiringProcessRedefined` **guard:** Only evolves when employer evaluation rules or job offer lifecycles are modified. **outcome:** Ensures the hiring flow remains compliant with domain-driven rules.
`DeckService`	Application	**event:** `SwipeGestureUpdated` **guard:** Changes when UI/UX interaction logic (e.g., animations or gesture thresholds) is modified. **outcome:** Affects visual representation and swipe mechanics, not core business logic. **note: Swiping is presentation logic, not business logic.**
`EventDispatcher`	Application	**event:** `EventRoutingChanged` **guard:** Changes when event propagation or inter-service communication models are redefined. **outcome:** Updates internal event broadcasting and subscription mechanisms.

Summary: Domain services evolve only when business policies change, while application services evolve with framework or presentation adjustments. This clear separation strengthens maintainability and aligns each module's purpose with its lifecycle of change.

Open/Closed Principle (OCP)

The system is **open for extension** but **closed for modification**, meaning new features can be added without altering existing code. This is achieved through abstraction and dependency injection.

Examples and Rationale:

- **Extension 1: Matching Algorithms event:** `NewAlgorithmIntroduced` **guard:** Implementations must conform to the `MatchingAlgorithm` interface. **outcome:** `AIMatchingAlgorithm` can be added without modifying `MatchingService`. **impact:** Existing matching logic remains stable and unaffected.
- **Extension 2: Deck Presentation event:** `SwipeAnimationAdded` **guard:** UI changes do not affect business logic. **outcome:** `DeckService` accepts a new animation strategy injected at runtime. **impact:** Domain layer remains isolated from visual logic.

These cases demonstrate how extensions are contained within the relevant layer, avoiding ripple effects and reinforcing architectural stability.

Liskov Substitution Principle (LSP)

Subtypes and implementations can substitute their base types without altering system behavior or violating domain expectations.

Examples:

- **User Role Substitution base type:** `UserProfile` **derived types:** `Recruiter`, `Employer` **guard:** Both maintain consistent interface contracts (`requestMatch()`, `updateProfile()`), differing only in context. **outcome:** Either can be used interchangeably in services expecting `UserProfile`, maintaining functional consistency.
- **Algorithm Substitution base type:** `MatchingAlgorithm` **derived types:** `BasicMatchingAlgorithm`, `AIMatchingAlgorithm` **guard:** All must produce a `MatchResult` structure with consistent semantics. **outcome:** Switching between algorithms does not require code changes in the `MatchingService`. **impact:** Reinforces predictability and polymorphic behavior.

By preserving substitutability, the design remains flexible without compromising domain integrity.

Interface Segregation Principle (ISP)

Interfaces are **focused, purpose-driven, and minimal**. This ensures clients depend only on the behaviors they actually use, avoiding "fat interfaces."

Examples:

- `MatchEvaluator` → Defines only scoring operations.
- `CandidateDataProvider` → Handles only data retrieval for profiles.
- `NotificationSender` → Responsible solely for delivering user notifications.

guard: Adding new methods requires creating a new interface, not modifying existing ones. **outcome:** Improves cohesion and testability by isolating roles per interface.

Dependency Inversion Principle (DIP)

High-level modules depend on abstractions, not concrete implementations. This design reduces coupling and increases flexibility when integrating new components.

Examples:

- **Dependency 1:** `MatchingService` depends on the abstraction `MatchingAlgorithm`, not its specific implementation. **event:** `AlgorithmInjected` **outcome:** Enables dynamic substitution (e.g., rule-based or AI-based strategies).
- **Dependency 2:** Application controllers depend on abstract domain interfaces rather than framework classes. **guard:** Domain layer remains independent of external frameworks. **outcome:** Domain logic can be tested or reused outside of any specific UI technology.

This inversion enforces stable architectural boundaries and supports long-term system evolution.

Example: Swiping and Matching Flow

```
class MatchingService {
  final SwipeRepository swipeRepository;
  final ConnectionService connectionService;

  MatchingService(this.swipeRepository, this.connectionService);

  Future<Result<void>> processSwipe(
      String userId, String profileId, SwipeDirection direction) async {
    await swipeRepository.save(userId, profileId, direction);

    if (direction == SwipeDirection.like) {
      final match = await checkForMatch(userId, profileId);
      if (match != null) {
        await connectionService.createConnection(match);
      }
    }
    return Result.success(null);
  }
}
```

This applies the **Single Responsibility Principle (SRP)**: the service only coordinates swiping logic and delegates persistence/connection creation to other components.

Example: Event RSVP Flow

```
Future<Result<Attendance>> rsvpEvent(String userId, String eventId) async {
  if (await eventRepository.hasCapacity(eventId)) {
    final attendance = Attendance(userId: userId, eventId: eventId, confirmed: true);
    await eventRepository.saveAttendance(attendance);
    return Result.success(attendance);
  }
  return Result.failure(CapacityReachedError());
}
```

This demonstrates the **Open/Closed Principle (OCP)**: RSVP logic can be extended (e.g., add waitlists) without changing its core behavior.

4.3. Life Cycle of Domain Objects

4.3.1. StudentProfile Life Cycle

The `StudentProfile` domain object represents a student within the system. Its lifecycle captures all possible states from creation to permanent deletion, including intermediate states such as active use and temporary archival. Understanding this lifecycle is critical to maintain data integrity, support aggregate consistency, and comply with domain rules.

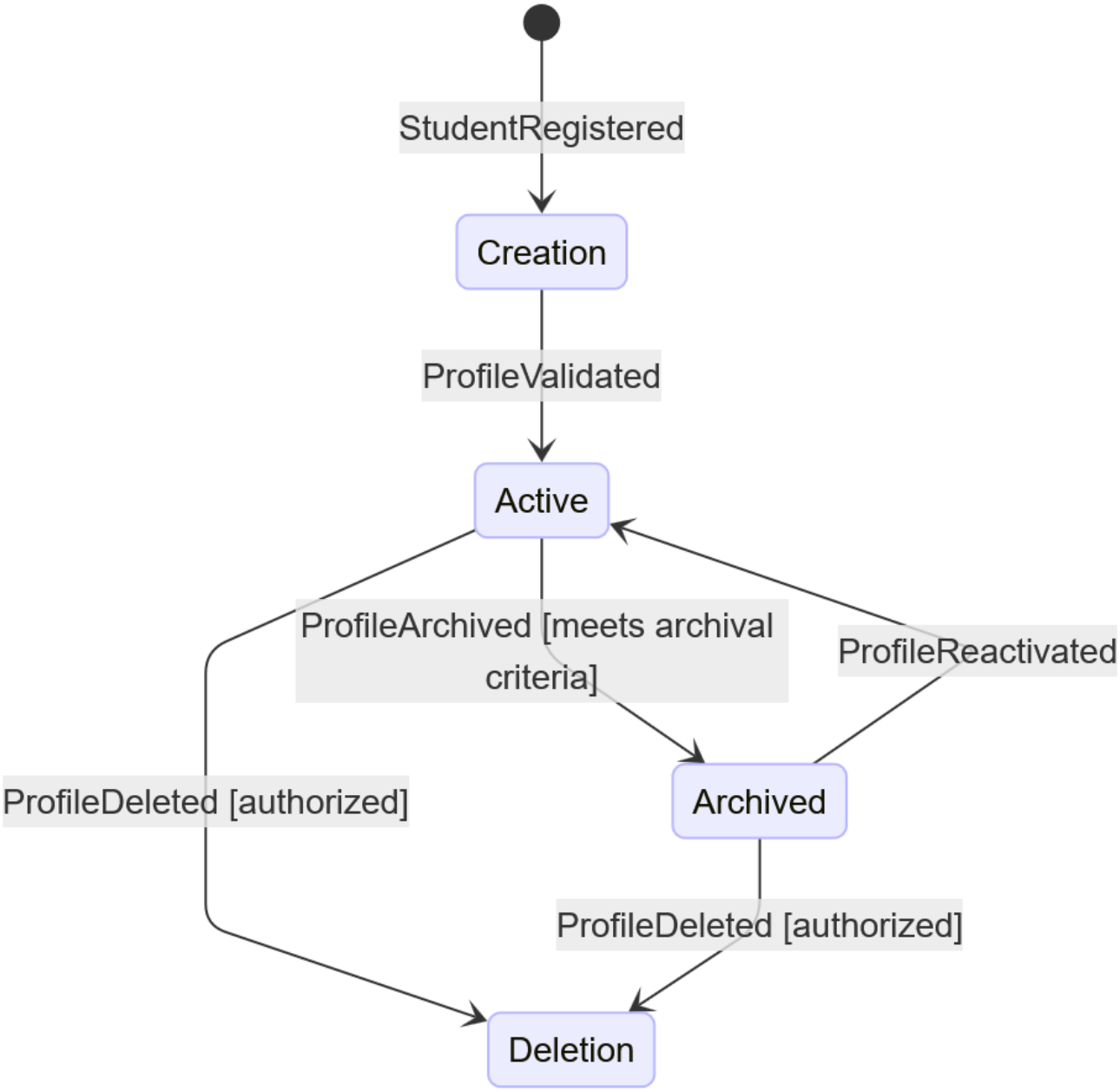
Lifecycle States

- **Creation**: triggered when a student registers or the system instantiates a profile; initial validation and default values are assigned
  - event: `StudentRegistered`
  - outcome: profile moves to **Active** after successful validation
- **Active**: default operational state of the profile; student can update personal information, add experiences, adjust visibility; interacts with other aggregates (e.g., enrollments, submissions)
  - events: `ProfileUpdated`, `ExperienceAdded`, `VisibilityChanged`
- **Archived**: temporary deactivation due to inactivity, graduation, or administrative action
  - profile becomes read-only
  - guard: only profiles meeting archival criteria can be archived
  - event: `ProfileArchived`
  - reactivation: `ProfileReactivated` moves profile back to **Active**
- **Deletion**: permanent removal from the system; ensures compliance and data integrity; may cascade to related aggregates (enrollments, submissions)
  - guard: only authorized or archived profiles may be deleted
  - event: `ProfileDeleted`

State Transitions

Transitions are implied within the state descriptions and governed by events and guard conditions. Explicit state changes include:

- Creation → Active: triggered by `StudentRegistered` and validation
- Active → Archived: triggered by `ProfileArchived` when archival criteria are met
- Archived → Active: triggered by `ProfileReactivated`
- Active/Archived → Deletion: triggered by `ProfileDeleted` when authorized



Aggregate Deletion Rules

The deletion of a `StudentProfile` must adhere to strict rules to maintain **aggregate consistency**, ensure **data integrity**, and comply with **domain policies**. The following rules apply:

- **Authorization Requirement**
  - A profile can only be deleted if it is either archived or explicitly authorized for deletion by an administrator.
  - Active profiles cannot be deleted without explicit authorization.
- **Cascading Deletions**
  - Deleting a `StudentProfile` may require cascading deletion or modification of related aggregates:
  - **Enrollments:** Any active or historical enrollments linked to the student must be removed or anonymized.

    - **Submissions/Assignments:** Student submissions may need to be archived, anonymized, or deleted according to retention policies.
    - **Notifications/Logs:** Events related to the profile should remain in the audit trail but must be decoupled from personally identifiable information.
- **Validation Checks**
    - Before deletion, the system must validate:
        - That no critical dependencies remain in other aggregates (e.g., group projects, team memberships).
        - That deletion will not violate invariants of related domain objects.
- **Compliance and Audit**
    - All deletion events must be logged with timestamp, actor, and reason for deletion.
    - Logs must be immutable and stored according to institutional compliance requirements.
    - This ensures traceability and supports future audits or investigations.
- **Reversibility**
    - Once deletion is executed, it is permanent.
    - If recovery is required, it must rely on backups or archived snapshots, not the active database.
- **Notification**
    - Dependent aggregates or stakeholders (e.g., course instructors, admin systems) may receive notifications when a profile is deleted, ensuring all systems maintain consistent state.
- **Guard Clauses**
    - Deletion is prevented if any of the above rules are violated.
    - Only profiles satisfying all deletion conditions are allowed to transition into the `Deletion` state.

4.4. Refactoring Towards Deeper Insight and Breakthroughs

4.4.1. Discovery Journal:

NOTE	The Discovery Journal captures the domain insights uncovered through refactoring, documenting how each change led to a deeper understanding of the model and improved its expressiveness.

This Discovery Journal records key domain insights and refactors that, during development, improved the clarity, consistency, and expressiveness of our domain model.

Entry 1: Address as immutable value object

- Context/Problem: Student profiles were storing address fields individually, leading to

inconsistent validation and duplicate address logic across forms.

- Domain Insight: Address is a value object in our domain—immutable, self-validating, and enforcing ISO-3166 alpha-2 country codes.
- Change Applied: Implemented Address VO (factory validation and normalization) and embedded it in StudentProfile with Supabase persistence and RLS.
- Impact: Single source of truth for address rules; fewer bugs and simpler forms within the Candidate/Student Profile context.
- Author/Date: Carlos Pepín — 2025-10-23
- Reference: https://github.com/uprm-inso4116-2025-2026-s1/semester-project-uprm_professional_portfolio/commit/cc540adb17bdfc56002f46ce6c89411821d15366

Entry 2: Chat service contract aligned across layers

- Context/Problem: Message shape differed between in-memory ChatService and Supabase-backed service, causing mismatches when switching implementations.
- Domain Insight: Message is a core domain concept; its shape must be stable across persistence to avoid leaking infrastructure details.
- Change Applied: Unified ChatMessage model and added explicit mapping in ChatServiceSupabase (body → text, server timestamps, null checks) with integration test.
- Impact: Reliable Message send/fetch for Recruiter□Candidate conversations and easier future refactors.
- Author/Date: Carlos Pepín — 2025-10-18
- Reference: https://github.com/uprm-inso4116-2025-2026-s1/semester-project-uprm_professional_portfolio/commit/506e7b568736c0d7acfbf4c52bba0e2f85676e7c

Entry 3: Conversation model clarified

- Context/Problem: It was unclear how Conversations should manage messages and expose read-only history.
- Domain Insight: Conversation aggregates Messages for two parties and should expose an immutable view of its timeline while providing controlled mutation methods.
- Change Applied: Defined Conversation with participants, add/remove message operations, and unmodifiable messages getter; set expectations for ordering.
- Impact: Cleaner boundaries for the Messaging context and predictable UI rendering for Recruiter and Candidate threads.
- Author/Date: PR merged by Julian Vivas — 2025-10-16
- Reference: https://github.com/uprm-inso4116-2025-2026-s1/semester-project-uprm_professional_portfolio/pull/189

Entry 4: Matches UI expresses decision actions

- Context/Problem: Recruiters lacked a focused place to accept/skip/star potential Candidates; information needed to be scoped to matching decisions.

- Domain Insight: A Match is a triage view over a Candidate; the decision surface should present only fields relevant to matching.
- Change Applied: Implemented Matches screen with accept/skip/star actions and a domain-specific info dialog for Candidate context.
- Impact: Supports Recruiter decision flow and sets the stage for persisting Match decisions.
- Author/Date: Samarys — 2025-10-23
- Reference: https://github.com/uprm-inso4116-2025-2026-s1/semester-project-uprm_professional_portfolio/commit/b7ddec4f2ee30d2c71c4f8ed47b8f00520a34c4f

Entry 5: Authentication behind a clean boundary

- Context/Problem: Moving from mock auth to Supabase risked coupling UI/controllers to provider details.
- Domain Insight: Identity is cross-cutting; the domain User should be provider-agnostic and accessed via an interface.
- Change Applied: Integrated Supabase Auth while preserving existing controllers and routing guard abstraction.
- Impact: Real login/signup with minimal churn to domain code; Messages and Matches now attach to authenticated users safely.
- Author/Date: Carlos Pepín — 2025-10-18
- Reference: https://github.com/uprm-inso4116-2025-2026-s1/semester-project-uprm_professional_portfolio/commit/dc9007c14e6ab86f53db8f3ae7e09682c461f8e0

Entry 6: Matches workflow merged as feature module

- Context/Problem: Recruiters lacked a consistent triage workflow for Candidate evaluation; actions were not standardized across the app.
- Domain Insight: Match is a dedicated workflow where Recruiters make Accept/Skip/Star decisions using a focused, minimal view of the Candidate.
- Change Applied: Merged the Matches feature with its screen/controllers and standardized decision actions for triage.
- Impact: Cohesive Recruiter experience and a clear place in the model for Match decisions to evolve toward persistence.
- Author/Date: Julian Vivas — 2025-10-23
- Reference: https://github.com/uprm-inso4116-2025-2026-s1/semester-project-uprm_professional_portfolio/pull/244

Entry 7: Establish Matches route and assets

- Context/Problem: Early prototype lacked a clear navigation entry and domain-scoped UI for the Matches bounded context.
- Domain Insight: Matches must be a first-class feature area separate from full profiles, exposing only decision-relevant data to Recruiters.

- Change Applied: Added Matches screen template, route, and assets to anchor subsequent domain behavior (accept/skip/star).
- Impact: Unblocked iteration on Recruiter triage and aligned ubiquitous language around "Match" across code and UI.
- Author/Date: Julian Vivas — 2025-10-22
- Reference: https://github.com/uprm-inso4116-2025-2026-s1/semester-project-uprm_professional_portfolio/pull/238

Entry 8: Supabase chat: fetch/send Message through clean API

- Context/Problem: Messaging relied on local mocks; moving to Supabase risked leaking database schema (body vs text, timestamps) into domain code.
- Domain Insight: Message behavior belongs to the Communication context and should be accessed through a stable service API.
- Change Applied: Implemented fetch/send operations in ChatServiceSupabase with mapping and server-issued timestamps; wired to domain-facing service.
- Impact: Recruiter–Candidate conversations can move between mock and live backends without touching UI/domain code.
- Author/Date: PR merged by Julian Vivas — 2025-10-19
- Reference: https://github.com/uprm-inso4116-2025-2026-s1/semester-project-uprm_professional_portfolio/pull/234

Entry 9: Supabase integration with router auth guard

- Context/Problem: Introducing real authentication threatened to tangle infrastructure concerns inside feature UIs.
- Domain Insight: Authentication sits at the boundary; domain features (Matches, Messages) should depend only on an abstract session.
- Change Applied: Integrated Supabase within app initialization and go_router guard, leaving domain models/controller contracts intact.
- Impact: Recruiter and Candidate sessions protect routes without leaking provider specifics into the domain.
- Author/Date: PR merged by Julian Vivas — 2025-10-17
- Reference: https://github.com/uprm-inso4116-2025-2026-s1/semester-project-uprm_professional_portfolio/pull/225

Entry 10: Message model consistency and JSON contract

- Context/Problem: Inconsistent constructors and JSON mapping led to fragile Message parsing and coupling to storage.
- Domain Insight: Message is a simple domain object—immutable data with explicit toJson/fromJson independent of any database.
- Change Applied: Normalized Message model with constructor first, clear fields, and

symmetrical JSON methods; tightened toString for debugging.

- Impact: Fewer parsing errors, clearer tests, and stable interface for Conversation and Chat services.
- Author/Date: PR merged by Julian Vivas — 2025-10-17
- Reference: https://github.com/uprm-inso4116-2025-2026-s1/semester-project-uprm_professional_portfolio/pull/188

Chapter 5. Log Book

Name	Section	Added/Modified
Jean P. I. Sanchez	5., 4.4.1., 3.2.1.	Added, Modified
Kiara N. Perez	1.3.2, 4.3, 2.1.2	Modified, Added
Alexa M. Zaragoza	4.2.1, 2.2.1, 2.1.4, 4.6.1	Added, Modified
Heribiell A. Rodríguez Cruz	2.1.2, 2.2.2	Modified, Added
Pedro Rodriguez Velez	1.5, 2.1.6	Modified, Added
Carlos J. Pepín Delgado	4.3.1	Added
Samarys Barreiro Melendez	2.1.5	Modified
Diego A. Pérez Gandarillas	2.3.1	Added
Julian Vivas	4.2.1	Modified
Fernando Castro Cancel	2.3.2., 2.2.3, 2.2.4, 2.2.5	Modified